Si ringrazia “newbieMac” che ha redatto il file per Edicolac64.com

IL COMMODORE 64 – informazioni tecniche e altro

INDICE

LA MAPPA DELLA MEMORIA DEL COMMODORE 64

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
0000	D6510	0	**6510 On-Chip Data-Direction Register** Registro di direzione dei dati
0001	R6510	1	**6510 On-Chip 8-bit Input/Output Register** Registro di ingresso/uscita a 8 bit
0002		2	**Non usata**
0003-0004	ADRAY1	3-4	**Jump Vector: Convert Floating-Integer** Vettore di salto: conversione da virgola mobile a intero
0005-0006	ADRAY2	5- 6	**Jump Vector: Convert Integer-Floating** Vettore di salto: conversione da intero a virgola mobile
0007	CHARAC	7	**Search Character** Ricerca di carattere
0008	ENDCHR	8	**Flag: Scan for Quote at End of String** Flag: Ricerca delle virgolette alla fine di una stringa
0009	TRMPOS	9	**Screen Column From Last TAB** Colonna dello schermo dopo l'ultima TAB
000A	VERCK	10	**Flag: 0 = Load, 1 = Verify** Flag: 0 = Caricamento, 1 = Verifica
000B	COUNT	11	**Input Buffer Pointer / No. of Subscripts** Puntatore al buffer d'ingresso / N. di indici nelle matrici
000C	DIMFLG	12	**Flag: Default Array DIMension** Flag: DIMensione delle matrici in assenza di diversa specifica
000D	VALTYP	13	**Data Type: $FRF = String, $00 = Numeric** Tipo di dato: $FF = Stringa, $00 = Numerico
000E	INTFLG	14	**Data Type: $80 = Integer, $00 = Floating** Tipo di dato: $80 = Intero, $00 = in virgola mobile
000F	GARBFL	15	**Flag: DATA scan / LIST quote / Garbage Coll** Flag: scansione delle DATA/comando LIST/'Raccoglitore di rifiuti'
0010	000F	16	**Flag: Subscript Ref / User Function Call** Flag: Riferimento a indice / Chiamata a funzione definita dall'tente
0011	INPFLG	17	**Floag: $00 = INPUT, $40 = GET, $98 = READ** Falg: $00 = INPUT, $40 = GET, $98 = READ
0012	TANSGN	18	**Flag: TAN sign / Comparison Result** Flag: segno di TAN / Risultato di un confronto
0013		19	**Flag: INPUT Prompt** Flag: Stringa abbinata ad una INPUT
0014-0015	LINNUM	20-21	**Temp: Integer Value** Provvisorio: valore intero
0016	TEMPPT	22	**Pointer: Temporary String Stack** Puntatore: Stack provvisiorio per le stringhe
0017-0018	LASTPT	23-24	**Last Temp String Address** Ultimo indirizzo provvisorio di stringa
0019-0021	TEMPST	25-33	**Stack for Temporary Strings** Stack per stringhe provvisorie
0022-0025	INDEX	34-37	**Utility Pointer Area** Area per puntatori di sistema
0026-002A	RESHO	38-42	**Floating-Point Product of Multiply** Risultato di moltiplicazione in virgola mobile
002B-002C	TXTAB	43-44	**Pointer: Start of BASIC Text** Puntatore: Inizio del testo in BASIC
002D-002E	VARTAB	45-46	**Pointer: Start of BASIC Variables** Puntatore: Inizio dell'area per le variabili BASIC
002F-0030	ARYTAB	47-48	**Pointer: Start of BASIC Array** Puntatore: Inizio dell'area per le matrici BASIC
0031-0032	STREND	49-50	**Pointer: End of BASIC Arrays (+1)** Puntatore: Termine area delle matrici BASIC (+1)
0033-0034	FRETOP	51-52	**Pointer: Bottom of String Storage** Puntatore: Fondo dell'area riservata alle stringhe
0035-0036	FRESPC	53-54	**Utility String Pointer** Puntatore a stringhe di utilità
0037-0038	MEMSIZ	55-56	**Pointer: Highest Address Used by BASIC** Puntatore: Indirizzo più alto usato dal BASIC
0039-003A	CURLIN	57-58	**Current BASIC Line Number** Attuale numero di linea BASIC
003B-003C	OLDLIN	59-60	**Previous BASIC Line Number**

			Numero di linea BASIC precedente

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
003D-003E	**OLDTXT**	**61-62**	**Pointer: BASIC Statement for CONT** Puntatore: per l'istruzione CONT del BASIC
003F-0040	**DATLIN**	**63-64**	**Current DATA Line Number** Attuale numero di linea per le DATA
0041-0042	**DATPTR**	**65-66**	**Pointer: Current DATA Item Address** Puntatore: Indirizzo del corrente elemento dei DATA
0043-0044	**INPPTR**	**67-68**	**Vector: INPUT Routine** Vettore: Routine d'immissione (INPUT)
0045-0046	**VARNAM**	**69-70**	**Current BASIC Variable Name** Attuale nome di variabile BASIC
0047-0048	**VARPNT**	**71-72**	**Pointer: Current BASIC Variable Data** Puntatore: Attuali dati variabili del BASIC
0049-004A	**FORPNT**	**73-74**	**Pointer: Index Variable for FOR/NEXT** Puntatore alla variabile "indice" per cicli FOR/NEXT
004B-0060		**75-96**	**Temp Pointer / Data Area** Puntatore temporaneo / Area Data
0061	**FACEXP**	**97**	**Floating Point Accumulator #1: Exponent** Accumulatore #1 per virgola mobile: Esponente
0062-0065	**FACHO**	**98-101**	**Floating Accum. #1: Mantissa** Accumulatore #1 per virgola mobile: Mantissa
0066	**FACSGN**	**102**	**Floating Accum. #1: Sign** Accumulatore #1 per virgola mobile: Segno
0067	**SGNFLG**	**103**	**Pointer: Series Evaluation Constant** Puntatore : Costante per valutazione di serie
0068	**BITS**	**104**	**Floating Accum. #1: Overflow Digit** Accumulatore #1 per virgola mobile: cifra in eccesso
0069	**ARGEXP**	**105**	**Floating Point Accum. #2: Exponent** Accumulatore #2 per virgola mobile: Esponente
006A-006D	**ARGHO**	**106-109**	**Floating Accum. #2: Mantissa** Accumulatore #2 per virgola mobile: Mantissa
006E	**ARGSGN**	**110**	**Floating Accum. #2: Sign** Accumulatore #2 per virgola mobile: Segno
006F	**ARISGN**	**111**	**Sign Comparison Result: Accum. #1 vs #2** Segno risultante dal confronto tra Accumulatore #1 e Accumulatore #2
0070	**FACOV**	**112**	**Floating Accum. #1: Low-Order (Rounding)** Accumulatore #1 per virgola mobile: Arrotondamento
0071-0072	**FBUFPT**	**113-114**	**Pointer: Cassette Buffer** Puntatore al buffer per le cassette
0073-008A	**CHRGET**	**115-138**	**Subroutine: Get Next Byte of BASIC Text** Subroutine: Estrae il prossimo byte di testo BASIC
0079	**CHRGOT**	**121**	**Entry to Get Same Byte of Text Again** Punto d'ingresso per estrarre nuovamente lo stesso byte di testo
007A-007B	**TXTPRT**	**122-123**	**Pointer: Current Byte of BASIC Text** Puntatore al corrente byte di testo BASIC
008B-008F	**RNDX**	**139-143**	**Floating RND Function Seed Value** "Seme" per la funzione RND in virgola mobile
0090	**STATUS**	**144**	**Kernal I/O Status Word: ST** Parola di stato per I/O del Kernal: ST
0091	**STKEY**	**145**	**Flag: STOP key / RVS key** Flag: tasto STOP / tasto RVS
0092	**SVXT**	**146**	**Timing Constant for Tape** Costante di temporizzazione per il registratore a cassette
0093	**VERCK**	**147**	**Flag: 0 = Load, 1 = Verify**
0094	**C3P0**	**148**	**Flag: Serial Bus – Output Char. Buffered** Flag: Bus seriale – Emissione di un carattere dal buffer
0095	**BSOUR**	**149**	**Buffered Character for Serial Bus** Deposito per il carattere da emettere sul bus seriale
0096	**SYNO**	**150**	**Cassette Sync No.** Numero di sincronizzazione per registratore a cassette
0097		**151**	**Temp Data Area** Area temporanea per dati
0098	**LDTND**	**152**	**No. of Open Files / Index to File Table** Numero di file aperti / Indice alla tabella dei file
0099	**DFLTN**	**153**	**Default Input Device (0)** Codice del normale dispositivo di immissione (0)
009A	**DFLTO**	**154**	**Default Output (CMD) Device (3)** Codice del normale dispositivo di emissione (3)
009B	**ORTY**	**155**	**Tape Character Parity** Parità dei caratteri per il registratore a cassette
009C	**DPSW**	**156**	**Flag: Tape Byte-Received**

			Flag: segnala il ricevimento di un byte proveniente dal registratore
009D	**MSGFLG**	**157**	**Flag: $80 = Direct Mode, $00 = Program** Flag : $80 = Modalità Diretta, $00 = Programma

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
009E	**PTR1**	**158**	**Tape Pass 1 Error Log** Segnale di errore nel 1° passaggio del nastro
009F	**PTR2**	**159**	**Tape Pass 2 Error Log** Segnale di errore nel 2° passaggio del nastro
00A0-00A2	**TIME**	**160-162**	**Real-Time Jiffy Clock (approx) 1/60 sec** Impulsi jiffy (circa 1/60 di secondo) dell'orologio in tempo reale
00A3-00A4		**163-164**	**Temp Data Area** Area temporanea per dati
00A5	**CNTDN**	**165**	**Cassette Sync Countdown** Contatore all'indietro per sincronizzazione del registratore
00A6	**BUFPTN**	**166**	**Pointer: Tape I/O Buffer** Puntatore al buffer di ingresso/uscita del registratore
00A7	**INBIT**	**167**	**RS-232 Input Bits / Cassette Temp** Porta seriale RS-232: Bit in ingresso / Registratore
00A8	**BITCI**	**168**	**RS-232 Input Bit Count / Cassette Temp** Porta seriale RS-232: Contatore di bit in ingresso / Registratore
00A9	**RINONE**	**169**	**RS-232 Flag: Check for Start Bit** Porta seriale RS-232: Flag per verifica sul bit di Start
00AA	**RIDATA**	**170**	**RS-232 Input Byte Buffer / Cassette Temp** Porta seriale RS-232: Buffer per il byte in ingresso / Registratore
00AB	**RIPRTY**	**171**	**RS-232 Input Parity / Cassette Short Cnt** Porta seriale RS-232: Parità in ingresso / Contatore breve per registratore
00AC-00AD	**SAL**	**172-173**	**Pointer: Tape Buffer / Screen Scrolling** Puntatore al buffer del registratore / Scorrimento dello schermo
00AE-00AF	**EAL**	**174-175**	**Tape End Addresses / End of Program** Indirizzo di fine nastro / Fine del programma
00B0-00B1	**CMPO**	**176-177**	**Tape Timing Constants** Costanti di temporizzazione per il registratore
00B2-00B3	**TAPE1**	**178-179**	**Pointer: Start of Tape Buffer** Puntatore all'inizio del buffer del registratore
00B4	**BITTS**	**180**	**RS-232 Out Bit Count / Cassette Temp** Porta seriale RS-232: Contatore di bit in uscita / Registratore
00B5	**NXTBIT**	**181**	**RS-232 Next Bit to Sent / Tape EOT Flag** Porta seriale RS-232: Prossimo bit da inviare / Flag di Fine Testo per il registratore
00B6	**RODATA**	**182**	**RS-232 Out Byte Buffer** Porta seriale RS-232: Buffer per il byte d'uscita
00B7	**FNLEN**	**183**	**Lenght of Current File Name** Lunghezza del corrente nome di file
00B8	**LA**	**184**	**Current Logical File Number** Numero logico assegnato al file corrente
00B9	**SA**	**185**	**Current Secondary Address** Attuale indirizzo secondario
00BA	**FA**	**186**	**Current Device Number** Numero dell'attuale dispositivo
00BB-00BC	**FNADR**	**187-188**	**Pointer: Current File Name** Puntatore al nome del file corrente
00BD	**ROPRTY**	**189**	**RS-232 Out Parity / Cassette Temp** Porta seriale RS-232: Parità in uscita / registratore
00BE	**FSBLK**	**190**	**Cassette Read / Write Block Count** Conteggio dei blocchi in Lettura/Scrittura sul registratore
00BF	**MYCH**	**191**	**Serial Word Buffer** Buffer per parola seriale
00C0	**CAS1**	**192**	**Tape Motor Interlock** Blocco del motore del registratore
00C1-00C2	**STAL**	**193-194**	**I/O Start Address** Indirizzo iniziale di ingresso/uscita
00C3-00C4	**MEMUSS**	**195-196**	**Tape Load Temps** Locazioni temporanee per il registratore
00C5	**LSTX**	**197**	**Current Key Pressed: CHR$(n) 0 = No Key** Tasto attualmente premutoi: CHR$(n) oppure 0 = Nessun tasto
00C6	**NDX**	**198**	**No. of Chars. in Keyboard Buffer (Queue)** Numero di caratteri nel Buffer della tastiera (Coda)
00C7	**RVS**	**199**	**Flag: Print Reserve Chars.-1 = Yes, 0 = Not Used** Flag: Visualizzazione caratteri inversi: -1=Si, 0=Non usato
00C8	**INDX**	**200**	**Pointer: End of Logical Line for INPUT** Puntatore: Fine "logica" di una linea in una INPUT
00C9-00CA	**LXSP**	**201-202**	**Cursor X-Y Pos. at Start of INPUT** Posizione X-Y del cursore all'inizio di una INPUT
00CB	**SFDX**	**203**	**Flag: Print Shifted Chars.** Flag: Visualizzazione caratteri con shift

00CC	BLNSW	204	**Cursor Blink enable: 0 = Flash Cursor** Abilitazione lampeggio del cursore: = = cursore lampeggiante
00CD	BLNCT	205	**Timer: Countdown to Toggle Cursor** Temporizzatore: contatore per passaggio acceso/spento del cursore
INDIRIZZO ESADECIMALE	**ETICHETTA**	**LOCAZIONE DECIMALE**	**DESCRIZIONE**
00CE	GDBLN	206	**Character Under Cursor** Carattere attualmente sotto il cursore
00CF	BLNON	207	**Flag: Last Cursor Blink On/Off** Flag: Ultimo lampeggio acceso/spento del cursore
00D0	CRSW	208	**Flag: INPUT or GET from Keyboard** Flag: INPUT o GET da tastiera
00D1-00D2	PNT	209-210	**Pointer: Current Screen Line Address** Puntatore: Attuale indirizzo della linea di schermo
00D3	PNTR	211	**Cursor Column on Current Line** Numero della colonna occupata dal cursore nell'attuale riga
00D4	QTSW	212	**Flag: Editor in Quote Mode, $00 = NO** Flag: Editor in modalità "virgolette", $00 = NO
00D5	LNMX	213	**Physical Screen Line Lenght** Lunghezza della riga "fisica" sullo schermo
00D6	TBLX	214	**Current Cursor Physical Line Number** Numero della linea fisica occupata dal cursore
00D7		215	**Temp Data Area** Area temporanea per dati
00D8	INSRT	216	**Flag: Insert Mode, >0 = #INSTs** Flag: Modo Insert attivo
00D9-00F2	LDTB1	217-242	**Screen Line Link Table / Editor Temps** Tabella di collegamento per le linee dello schermo / Loacioni temporanee
00F3-00F4	USER	243-244	**Pointer: Current Screen Color RAM loc.** Puntatore: locazione RAM dell'attuale schermo a colori
00F5-00F6	KEYTAB	245-246	**Vector: Keyboard Decode Table** Vettore: Tabella di decodifica per la tastiera
00F7-00F8	RIBUF	247-248	**RS-232 Input Buffer Pointer** Puntatore al buffer d'ingresso per RS-232
00F9-00FA	ROBUF	249-250	**RS-232 Output Buffer Pointer** Puntatore al buffer di uscita per RS-232
00FB-00FE	FREKZP	251-254	**Free 0-Page Space for User Programs** Spazio libero in pagina 0 per programmi utente
00FF	BASZPT	255	**BASIC Temp Data Area** Area dei dati temporanei per il BASIC
0100-01FF		256-511	**Micro-Processor System Stack Area** Area per lo stack di sistema del microprocessore
0100-010A		256-266	**Floating to String Work Area** Area di lavoro utilizzata per numeri in virgola mobile e stringhe
0100-013E	BAD	256-318	**Tape Input Error Log** Segnalatore di errore di lettura su registratore
0200-0258	BUF	512-600	**System INPUT Buffer** Buffer di sistema per la INPUT
0259-0262	LAT	601-610	**KERNAL Table: Active Logical File No's** Tabella KERNAL: Numeri dei file logoco attivi
0263-026C	FAT	611-620	**KERNAL Table: Device No. for Each File** Tabella KERNAL: Numero del dispositivo abbinato a ciascun file
026D-0276	SAT	621-630	**KERNAL Table: Second Address Each File** Tabella KERNAL: Indirizzo secondario di ciascun file
0277-0280	KEYD	631-640	**Keyboard Buffer Queue (FIFO)** Coda per il buffer di tastiera (FIFO)
0281-0282	MEMSTR	641-642	**Pointer: Bottom of Memory for O.S.** Puntatore: Indirizzo più basso di memoria per il sistema operativo
0283-0284	MAMSIZ	643-644	**Pointer: Top of Memory for O.S.** Puntatore: indirizzo di memoria più alto per il sistema operativo
0285	TIMOUT	645	**Flag: Kernal Variable for IEEE Timeout** Flag: Variabile del Kernal per segnalare il tempo scaduto sulla linea IEEE
0286	COLOR	646	**Current Character Color Code** Codice per il colore del carattere corrente
0287	GDCOL	647	**Background Color Under Cursor** Cole dello sfondo sotto il cursore
0287	HIBASE	648	**Top of Screen Memory (Page)** Indirizzo più alto della memoria video (valore della pagina)
0289	XMAX	649	**Size of Keyboard Buffer** Dimensioni del buffer per la tastiera
028A	RPTFLG	650	**Flag: REPEAT Key Used, $80 = Repeat** Flag: Ripetizione attiva dei tasti usati
028B	KOUNT	651	**Repeat Speed Counter** Contatore di velocità per la ripetizione dei tasti
028C	DELAY	652	**Repeat Delay Counter** Contatore per l'attesa nella ripetizione dei tasti
028D	SHFLAG	653	**Flag: Keyboard SHIFT Key/CTRL Key/C = Key**

			Flag: tasti SHIFT, CTRL e C = attivi
028E	**LSTSHF**	**654**	**Last Keyboard Shift Pattern** Ultima configurazione di shift per la tastiera
INDIRIZZO ESADECIMALE	**ETICHETTA**	**LOCAZIONE DECIMALE**	**DESCRIZIONE**
028F-0290	**KEYLOG**	**655-656**	**Vector: Keyboard Shift Pattern** Vettore: Configurazione di shift per la tastiera
0291	**MODE**	**657**	**Flag: $00 = Disable SHIFT Keys, $80 = Enable** Flag di abilitazione/disabilitazione dei tasti SHIFT; $00=disattivi, $80=attivi
0292	**AUTODN**	**658**	**Flag: Auto Scroll Down, 0 = On** Flag: Scroll automatico verso il basso, 0=attivo
0293	**M51CTR**	**659**	**RS-232: 6551 Control Register Image** RS-232: Immagine del registro di controllo del 6551
0294	**M51CDR**	**660**	**RS-232: 6551 Command Register Image** RS-232: Immagine del registro di comando del 6551
0295-0296	**M51AJB**	**661-662**	**RS-232: Non Standard BPS (Time/2-100) USA** RS-232: Velocità non standard
0297	**RSSTAT**	**663**	**RS-232: 6551 Status Register Image** RS-232: Immagine del registro di stato del 6551
0298	**BITNUM**	**664**	**RS-232: Number of Bits Left to Send** RS-232: Numero di bit ancora da trasmettere
0299-029A	**BAUDOF**	**665-666**	**RS-232 Baud Rate: Full Bit Time (µs)** RS-232, velocità in Baud: Intervallo di tempo per ogni bit (in µs)
029B	**RIDBE**	**667**	**RS-232 Index to End of Input Buffer** RS-232: Indice per la fine del buffer d'immissione
029C	**RIDBS**	**668**	**RS-232 Start of Input Buffer (Page)** RS-232: Inizio del buffer d'immissione (valore della pagina)
029D	**RODBS**	**669**	**RS-232 Start of Output Buffer (Page)** RS-232: Inizio del buffer di uscita (valore della pagina)
029E	**RODBE**	**670**	**RS-232 Index to End of Output Buffer** RS-232: Indice per la fine del buffer d'uscita
029F-02A0	**IRQTMP**	**671-672**	**Holds IRQ vector During Tape I/O** Contiene il vettore IRQ durante le operazioni di I/O su registratore
02A1	**ENABL**	**673**	**RS-232 Enables** Abilitazione della porta RS-232
02A2		**674**	**TOD Sense During Cassette I/O** Segnalatore di TOD durante le operazioni di I/O su nastro
02A3		**675**	**Temp Storage for Cassette Read** Deposito temporaneo per la lettura da nastro
02A4		**676**	**Temp D1IRQ Indicator for Cassette Read** Deposito temporaneo per l'indicatore D1IRQ durante la lettura da nastro
02A5		**677**	**Temp for Line Index** Deposito temporaneo per l'indice di linea
02A6		**678**	**PAL/NTSC Flag, 0 = NTSC, 1 = PAL** Flag per la modalità PAL/NTSC, 0=NTSC, 1=PAL
02A7-02FF		**679-767**	**Unused** Locazioni non utilizzate
0300-0301	**IERROR**	**768-769**	**Vector: Print BASIC Error Message** Vettore: Visualizzazione dei messaggi d'errore del BASIC
0302-0303	**IMAIN**	**770-771**	**Vector: BASIC Warm Start** Vettore: Inizializzazione "a caldo" del BASIC
0304-0305	**ICRNCH**	**772-773**	**Vector: Tokenize BASIC Text** Vettore: Conversione in token del testo BASIC
0306-0307	**IQPLOP**	**774-775**	**Vector: BASIC Text LIST** Vettore: LIST del testo BASIC
0308-0309	**IGONE**	**776-777**	**Vector: BASIC Char. Dispatch** Vettore: Emissione di caratteri BASIC
030A-030B	**IEVAL**	**778-779**	**Vector: BASIC Token Evaluation** Vettore: Valutazione dei token del BASIC
030C	**SAREG**	**180**	**Storage for 6502 .A register** Deposito per il registro .A del 6502
030D	**SXREG**	**181**	**Storage for 6502 .X Register** Deposito per il registro .X del 6502
030E	**SYREG**	**782**	**Storage for 6502 .Y Register** Deposito per il registro .Y del 6502
030F	**SPREG**	**783**	**Storage for 6502 .SP Register** Deposito per il registro .SP del 6502
0310	**USRPOK**	**784**	**USR Function Jump Instr (4C)** Istruzione di salto (4C) per la funzione USR
0311-0312	**USRADD**	**785-786**	**USR Address Low Byte/High Byte** Indirizzo nel formato lo/hi della routine USR
0313		**787**	**Unused** Locazione non usata
0314-0315	**CINV**	**788-789**	**Vector: Hardware IRQ Interrupt** Vettore di interrupt hardware IRQ
0316-0317	**CBINV**	**790-791**	**Vector: BRK Instr. Interrupt** Vettore: Interrupt a seguito di BRK

0318-0319	**NMINV**	**792-793**	**Vector: Non-Maskable Interrupt** Vettore Interrupt non mascherabile
INDIRIZZO ESADECIMALE	**ETICHETTA**	**LOCAZIONE DECIMALE**	**DESCRIZIONE**
031a-031b	**iopen**	**794-795**	**KERNAL OPEN Routine Vector** Vettore per routine OPEN del Kernal
031C-031D	**ICLOSE**	**796-797**	**KERNAL CLOSE Routine Vector** Vettore per routine CLOSE del Kernal
031E-031F	**ICHKIN**	**798-799**	**KERNAL CHKIN Routine Vector** Vettore per routine CHKIN del Kernal
0320-0321	**ICHKOUT**	**800-801**	**KERNAL CHKOUT Routine Vector** Vettore per routine CHKOUT del Kernal
0322-0323	**ICLRCH**	**802-803**	**KERNAL CLRCHN Routine Vector** Vettore per routine CLRCHN del Kernal
0324-0325	**IBASIN**	**804-805**	**KERNAL CHRIN Routine Vector** Vettore per routine CHRIN del Kernal
0326-0327	**IBSOUT**	**806-807**	**KERNAL CHROUT Routine Vector** Vettore per routine CHROUT del Kernal
0328-0329	**ISTOP**	**808-809**	**KERNAL STOP Routine Vector** vETTORE PER ROUTINE stop DEL kERNAL
032A-032B	**IGETIN**	**810-811**	**KERNAL GETIN Routine Vector** Vettore per routine GETIN del Kernal
032C-032D	**ICLALL**	**812-813**	**KERNAL CLALL Routine Vector** Vettore per routine CLALL del Kernal
032E-032F	**USRCMD**	**814-815**	**User-Defined Vector** Vettore definito da utente
0330-0331	**ILOAD**	**816-817**	**KERNAL LOAD Routine Vector** Vettore per routine LOAD del Kernal
0332-0333	**ISAVE**	**818-819**	**KERNAL SAVE Routine Vector** Vettore per routine SAVE del Kernal
0334-033B		**820-827**	**Unused** Locazioni non utilizzate
033C-03FB	**TBUFFER**	**828-1019**	**Tape I/O Buffer** Memoria tampone per operazioni di I/O su registratore
03FC-03FF		**1020-1023**	**Unused** Locazioni non utilizzate
0400-07FF	**VICSCN**	**1024-2047**	**Sprite Data Pointers** Puntatori ai dati per gli sprite
07F8-7FFF		**2048-40959**	**Normal BASIC Program Space** Area normalmente riservata ai programmi BASIC
8000-9FFF		**32768-40959**	**VSP Cartridge ROM – 8192 Bytes** Area riservata alle cartucce ROM (8192 byte)
A000-BFFF		**40960-49151**	**BASIC ROM – 8192 Bytes (or 8K RAM)** Area riservata alle ROM del Basic (circa 8 Kbyte)
C000-CFFF		**49152-53247**	**RAM – 4096 Bytes** Area RAM – 4096 byte
D000-DFFF		**53248-57343**	**Input/Output Devices and Color RAM or Char. Generator ROM** Area riservata ai dispositivi di I/O ed alla RAM del colore, oppure al generatore di caratteri ROM
E000-FFFF		**57344-65535**	**KERNAL ROM – 8192 Bytes (or 8K RAM)** ROM del Kernal – 8192 byte (o 8 Kbyte di RAM)

Il SO del Commodore 64

Panoramica sugli aspetti utili o insoliti del sistema operativo del C64

La prima cosa che si deve conoscere di un computer, ai fini della programmazione, è la configurazione globale della sua mappa di memoria. Nel caso del Commodore 64, il microprocessore 6510, estremamente versatile, può “vedere” una tra numerose mappe possibili a seconda del modo in cui viene configurato. Infatti il 6510 può configurare differenti banchi di memoria: la selezione può essere realizzata fisicamente, applicando una tensione di 5V o di 0V sui piedini 8 e 9 della porta d’espansione, oppure via software modificando il contenuto dell’indirizzo 1. Entrambe le operazioni modificano radicalmente il modo in cui il 6510 osserva la memoria. La figura rappresenta la normale mappa di memoria del Commodore 64 ed i 64 Kbyte ai quali il 6510 accede in condizioni normali. La tabella contiene una minuziosa descrizione della mappa di memoria.

Uno sguardo alla ROM
Poiché il processore 6510 usa indirizzi a 16 bit, può indirizzare un massimo di 65536 locazioni di memoria, ossia 64 Kbyte. Il Commodore possiede in totale 84 Kbyte suddivisi tra ROM e RAM, ma è in grado di “vedere” solo un blocco di 64 Kbyte alla volta, sostituendo i banchi delle aree di memoria all’interno della sua mappa.
Il diagramma in figura mostra la memoria normalmente “vista” dal 6510. Per poter accedere alle aree nascoste, occorre selezionare speciali registri di sostituzione. Ad esempio: se un programma è scritto completamente in codice macchina, la ROM BASIC non è necessaria e può essere eliminata dalla mappa rendendo disponibile la RAM posta sotto di essa.

Supponiamo che la memoria del Commodore si trovi nello stato normale (di ‘default’). La prima cosa da osservare è che il computer possiede 64 Kbyte di RAM e 20 Kbyte di ROM. Poiché in nessun caso il 6510 può indirizzare più di 64 Kbyte degli 84 presenti, appare evidente la necessità di una funzione che permetta di sele-zionare i banchi di memoria. Il diagramma della mappa qui sopra, mostra che la ROM dell’interprete BASIC, ossia il programma in codice macchina che esegue i programmi in BASIC, è compresa tra gli indirizzi $A000 e $CFFF, mentre la ROM del Kernel, ossia il codice che gestisce tutte le funzioni di input/output, si trova tra le locazioni $E000 e $FFFF.
Sotto questi due blocchi ROM ci sono due aree RAM, ciascuna di 8 Kbyte, che, in condizioni normali, il 6510 non può utilizzare re (sebbene istruzioni POKE e STA, relative agli indirizzi di queste aree, vi filtrino effettivamente). Un’istruzione PEEK fornisce il contenuto della ROM: la RAM può essere letta solo escludendo, dalla mappa di memoria, la ROM corrispondente. Si rammenti tuttavia che, qualora si vogliano usare i comandi PEEK e POKE del BASIC, non si può certo escludere la ROM del BASIC!

CONFIGURAZIONE DELLA MAPPA DI MEMORIA DEL COMMODORE 64

Locazioni	Funzione
$0000-$0001	Registri di controllo della mappa di memoria del 6510
$0002-$03FF	RAM del sistema operativo
$0400-$07F7	RAM del video
$07F8-$07FF	Puntatori agli sprite
$0800-$9FFF	Area dei programmi BASIC per l’utente (variabili comprese)
$A000-$BFFF	ROM dell’interprete BASIC
$C000-$CFFF	RAM libera (4 Kbyte)
$D000-$D02E	Registri di controllo del chip VIC II (6566/9)
$D02F-$D3FF	Ripete le immagini del VIC II
$D400-$D41C	Registri di controllo del chip SID (6581)
$D41D-$D7FF	Ripete le immagini del SID
$D800-$DBE7	Nybble del colore (4 bit ciascuno)
$DBE8-$DBFF	Nybble non utilizzati
$DC00-$DC0F	Registri di controllo del chip di I/O CIA #1 (6526)
$DC10-$DCFF	Ripete le immagini del CIA #1
$DD00-$DD0F	Registri di controllo del chip di I/O CIA #2 (6526)
$DD10-$DDFF	Ripete le immagini del CIA #2
$E000-$FFFF	ROM Kernel

Come i programmi BASIC necessitano di un'area RAM in cui registrare le proprie variabili, così le routine dell'interprete e del Kernel ricorrono alla RAM del SO; questa è compresa tra gli indirizzi $0002 e $03FF ma qui si trova anche lo stack, situato tra le locazioni$0100 e $01FF.
La RAM del SO si trova in pagina zero poiché questa è la parte della mappa di memoria che consente un accesso più rapido alla RAM, fattore essenziale per l'attività del sistema operativo. Se si desidera abbinare un programma BASIC ad uno in codice macchina, è necessario conoscere bene il funzionamento di quest'area di RAM.
Esaminiamo, adesso, due importanti aree della RAM del SO: i puntatori del BASIC ed i vettori del codice macchina.

Puntatori del BASIC

In alcune circostanze può essere utile modificare il contenuto dei puntatori del BASIC. Per esempio: gli indirizzi $002B (43 in decimale) e $002C (44), abitualmente contengono i valori 1 e 8. Questo è l'indirizzo iniziale del BASIC (nel formato lo-hi o "basso-alto") e significa che il BASIC inizia dalla locazione 8x256 + 1 = 2049, ossia $0801. In realtà il BASIC comincia alla locazione $0800 ma, dato che il sistema operativo esige che il primo byte sia sempre 0, il programma BASIC inizia effettivamente da $0801 (detto per inciso, $0801 è anche l'indirizzo da cui parte il salvataggio di un programma BASIC quando si usa il programma monitor per il codice macchina del Commodore).

TABELLA BASIC DEL	DEI PUNTATORI AL **CBM 64**
Locazioni	**Funzione**
$002B-$002C	Inizio area BASIC per l'utente
$002D-$002E	Inizio area variabili BASIC
$002F-$0030	Inizio area matrici BASIC
$0031-$0032	Fine area matrici BASIC + 1
$0033-$0034	Fondo dell'area stringhe
$0035-$0036	Utility
$0037-$0038	Fine area BASIC per l'utente

Prima di caricare un programma BASIC, è possibile modificare l'indirizzo iniziale del BASIC intervenendo su questi puntatori. Occorre però avere l'accortezza di far iniziale il BASIC ad un margine di pagina (una "pagina" rappresenta un blocco di memoria di 256 byte), cioè che la locazione $002B contenga il valore 1 e che il primo byte sia uno 0. Quindi la linea:

```
POKE 2560,0:POKE 44,10:NEW
```

eseguita in modo diretto, innalza di due pagine il fondo della memoria portandolo all'indirizzo $2560; l'istruzione NEW permette di azzerare con grande rapidità tutti i puntatori (contenuti nelle locazioni da $002D a $0038). L'innalzamento del fondo del BASIC, permette di avere in memoria due programmi BASIC contemporaneamente: la procedura prevede il caricamento del primo programma, l'innalzamento del fondo e BASIC e, infine, il caricamento del secondo programma.
Il seguente intervento sui puntatori, usato con maggior frequenza del precedente, permette di abbassare la sommità della memoria per fare spazio ad un programma in codice macchina. La linea:

```
POKE 56,159:POKE 51,0:POKE 52,159
```

abbassa la sommità della memoria di una pagina. Dopo aver rimosso dall'area di memoria del BASIC un blocco di RAM, si può star sicuri che il SO non userà quest'area per registrarvi le variabili BASIC. Questa precauzione mette al sicuro il cosice macchina da accidentali danneggiamenti!

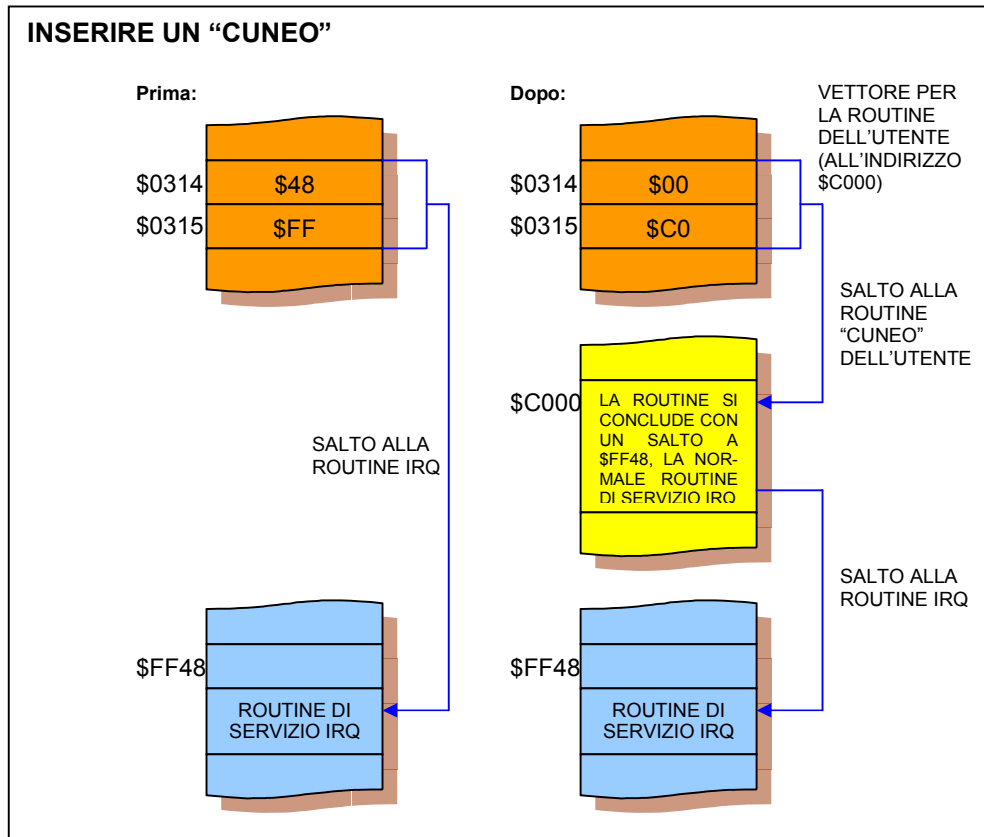
I vettori RAM

Un secondo blocco della RAM, di particolare interesse per i programmatori in linguaggio macchina, contiene i vettori RAM ed è compreso tra le locazioni $0314 e $0333. Un vettore RAM è paragonabile ad un comune scambio ferroviario; quando un treno (o, per analogia, il processore 6510 che esegue il proprio programma) attraversa uno scambio, lo supera senza cambiare direzione di marcia. Tuttavia, in alcuni casi, può essere necessario deviarlo su di una linea laterale, facendogli attraversare una o due stazioni secondarie, prima di riportarlo su quella principale.
Prendiamo ad esempio il vettore IRQ (Interrupt ReQuest). Quando il Commodore 64 funziona normalmente, ogni sessantesimo di secondo viene avviato uno dei temporizzatori del chip di I/O del 6526 che pone a livello basso la linea IRQ del 6510. Terminata l'elaborazione dell'istruzione corrente, il 6510 risponde all' "abbassamento" della linea IRQ, generando una interrupt ed eseguendo

TABELLA CBM 64	**DEI VETTORI DEL**
Locazioni	**Vettore**
$0314-$0315	IRQ (Interrupt)
$0316-$0317	BRK (Interrupt)
$0318-$0319	NMI (Interrupt)
$031A-$031B	Routine OPEN del Kernel
$031C-$031D	Routine CLOSE del Kernel
$031E-$031F	Routine CHKIN del Kernel
$0320-$0321	Routine CHKOUT del Kernel
$0322-$0323	Routine CLRCHN del Kernel
$0324-$0325	Routine CHRin del Kernel
$0326-$0327	Routine CHROUT del Kernel
$0328-$0329	Routine STOP del Kernel
$032A-$032B	Routine GETIN del Kernel
$032C-$032D	Routine CLALL del Kernel
$032E-$032F	Definito dall'utente
$0330-$0331	Routine LOAD del Kernel
$0332-$0333	Routine SAVE del Kernel

una routine di servizio che inizia all'indirizzo $FF48. Questa routine, per le IRQ, esamina anche la tastiera per individuare eventuali pressioni di tasti da parte dell'utente. Una delle prime istruzioni eseguite è:

```
JMP ($3014)
```

che provoca un salto indiretto all'indirizzo contenuto nelle locazioni $0314 (byte basso) e $0315 (byte alto) che si trovano in RAM. In questo modo, modificando il loro contenuto, sarà possibile puntare ad una diversa sezione del programma. Elaborata questa sezione il microprocessore potrà essere inviato all'indirizzo registrato in origine nelle due locazioni. Con questo sistema al 6510 viene fatta eseguire, ogni sessantesimo di secondo, una particolare sezione di codice scritta dall'utente (purché questa non sia troppo lunga, nel qual caso impedirebbe l'uso di qualsiasi routine Kernel). Una routine così concepita, viene chiamats "cuneo" per il modo in cui si "incunea" nel normale funzionamento del sistema.

"Cunei" di tempo
Il processore 6510 viene interrotto ogni sessantesimo di secondo per eseguire operazioni di servizio quali la scansione della tastiera. L'indirizzo iniziale della routine di servizio si trova alle locazioni $0314 e $0315.
Modificando l'indirizzo, contenuto in questa coppia di locazioni, è possibile "incuneare" una breve sezione di programma scritta, dall'utente, che viene eseguita prima della routine di servizio. Di solito, infatti, il controllo viene ripassato alla normale routine di servizio al termine della routine detta "cuneo". Pertanto il sistema esegue la routine utente ogni qualvolta venga generato questo tipo di interrupt, ossia ogni sessantesimo di secondo.

L'OROLOGIO DEL COMMODORE

Questo programma "caricatore" in BASIC ed il successivo listato assembly, illustrano come sia possibile inserire una sezione di programma "cuneo" in una normale routine di gestione degli interrupt. Il "cuneo" in questione visualizza un orologio in un angolo dello schermo. Poiché l'ora viene aggiornata dal sistema ricorrendo alla normale interrupt generata ogni sessantesimo di secondo, la visualizzazione dell'orologio avverrà anche durante l'inserimento ed esecuzione, in contemporanea, di un programma BASIC. Per prima cosa la routine "rimette l'ora" con i tempi iniziali che le vengono comunicati, poi modifica il contenuto di $0314 e $0315 per puntare alla routine "cuneo" che aggiornerà e visualizzerà l'orologio ogni volta che verrà generato un IRQ. Come operazione finale, esegue un salto (JMP) alla normale routine di sistema IRQ il cui indirizzo viene conservato nelle locazioni VETT e VETT+1.
Come alternativa all'introduzione ed all'assemblaggio di questo programma "sorgente", è possibile inserire il programma sotto forma di istruzioni DATA che contengono il codice macchina. L'esecuzione del programma "caricatore" in BASIC, produce la comparsa dell'orologio nell'angolo superiore destro dello schermo. Il "checksum" alla linea 1420 visualizza un errore qualora uno dei valori nelle DATA non risultasse trascritto giusto.
L'orario iniziale viene selezionato con una serie di istruzioni POKE introducendo le ore, i minuti ed i secondi, rispettivamente, nelle locazioni 50129, 50130 e 50131. Alla fine del programma di caricamento, le linee da 1480 in poi avviano l'orologio a partire dall'ora pomeridiana 04:30:00. Per modificare questo orario basta modificare i valori registrati nelle tre locazioni. SYS 50138 lancia il programma in codice macchina ed avvia l'orologio; questo può essere disattivato premendo contemporaneamente i tasti RUN/STOP e RESTORE o impartendo il comando SYS 50237 che lancia un'apposita routine in codice macchina acclusa al programma.

Programma di caricamento BASIC

```
1000 REM ** CARICATORE BASIC DELL'OROLOGIO **
1010 DATA 173,166,2,240,10,169,128,13,14
1020 DATA 221,141,14,221,48,8,169,127,45
1030 DATA 14,221,141,14,221,169,127,45,
1040 DATA 15,221,141,15,221,173,208,195
1050 DATA 41,128,141,208,195,173,209,195
1060 DATA 32,28,197,13,208,195,141,11
1070 DATA 221,173,210,195,32,28,197,141
1080 DATA 10,221,173,211,195,32,28,197
1090 DATA 141,9,221,169,0,141,8,221,120
1100 DATA 173,20,3,141,214,195,173,21,3
1110 DATA 141,215,195,169,76,141,20,3
1120 DATA 169,196,141,21,3,88,96,120,173
1130 DATA 214,195,141,20,3,173,215,195
1140 DATA 141,21,3,88,96,193,216,195,201
1150 DATA 6,240,3,76,8,197,169,255,141
1160 DATA 216,195,173,213,195,240,243
1170 DATA 193,11,221,190,41,128,208,5
1180 DATA 169,1,76,111,196,169,16,141,68
1190 DATA 4,173,212,195,141,38,216,169
1200 DATA 13,141,39,4,173,212,195,141,39
1210 DATA 216,138,41,16,32,14,197,141,28
1220 DATA 4,173,212,195,141,28,216,138
1230 DATA 32,22,197,141,29,4,193,212,195
1240 DATA 141,29,216,169,58,141,30,4,173
1250 DATA 212,195,141,30,216,173,10,221
1260 DATA 170,32,14,197,141,31,4,193,212
1270 DATA 195,141,31,216,138,32,22,197
1280 DATA 141,32,4,173,212,195,141,32
1290 DATA 216,169,47,141,33,4,193,212
1300 DATA 195,141,33,216,173,9,221,170
1310 DATA 32,14,197,141,34,4,193,212,195
1320 DATA 141,34,216,138,32,22,197,141
1330 DATA 35,4,173,212,195,141,35,216
1340 DATA 169,46,141,36,4,173,212,495
1350 DATA 141,36,216,173,8,221,105,48
1360 DATA 141,37,4,193,212,195,141,37
1370 DATA 216,238,216,195,108,214,195,74
1380 DATA 74,74,74,24,105,48,96,41,15,24
1390 DATA 105,48,96,160,255,56,200,233
1400 DATA 10,176,251,105,10,141,217,195
1410 DATA 152,10,10,10,10,13,217,195,96
1420 DATA 42131: REM * CHECKSUM *
1430 CC=0
1440 FOR I=50138 TO 50481
1450 READ X: CC=CC+1: POKE I,X
1460 NEXT
1470 READ X: IF CC<>X THEN PRINT
        "CHECKSUM ERROR"
1480 REM * COLLAUDO DELL'OROLOGIO *
1490 POKE 50128,128: REM AM/PM
1500 POKE 50132,8: REM COLORE
1510 POKE 50133,1: REM SCHERMO
1520 POKE 50129,4: REM ORE
1530 POKE 50130,30: REM MINUTI
1540 POKE 50131,0: REM SECONDI
1550 SYS 50138: REM CHIAMA LA ROUTINE
```

Usare SYS 50237, nel modo diretto o nel programma, per aggiungere o eliminare la routine di “cuneo”.

Listato Assembly

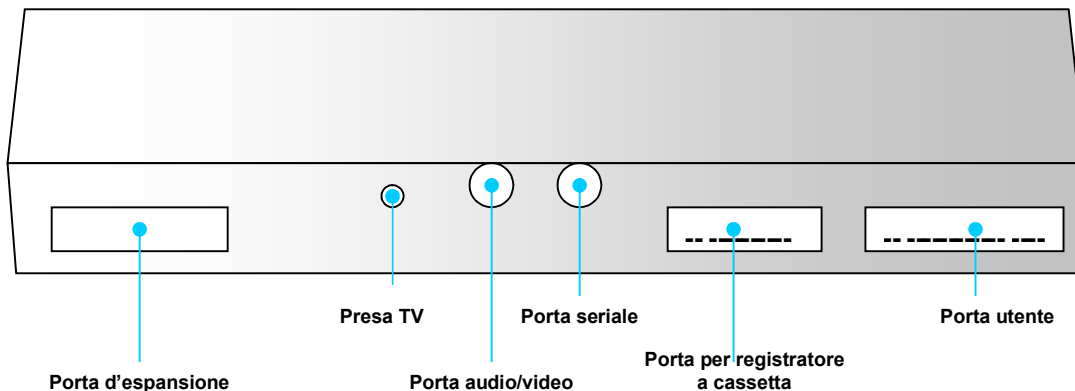
```
; ***************************************************
; *                                                 *
; *        PROGRAMMA CUNEO PER                      *
; *             OROLOGIO IRQ                        *
; *                                                 *
; *   50128 = AM=0 / PM=129                         *
; *   50129 = ORE                                   *
; *   50130 = MINUTI                                *
; *   50131 = SECONDI                               *
; *   50132 = COLORE OROLOGIO                       *
; *   50133 = DISPLAY ON=1/OFF=0                    *
; *                                                 *
; *     INSERIMENTO = SYS 50138                     *
; *     RIMOZIONE = SYS 50237                       *
; *                                                 *
; ***************************************************

IRQVET = $0314          ; vettore RAM di IRQ
CLOCK = $DD08           ; registro TOD (ora del giorno)
D2CRA = $DD0E           ; VIA#2 CRA
D2CRB  = $DD0F          ; VIA#2 CRB
PALNTS = $02A6          ; Flag PAL/NTSC
RITMO = $06             ; un display ogni 6 IRQ
CIFRA = $30             ; codice schermo per '0'
PUNTO = $2E             ; codice schermo per '.'
BARRA = $2F             ; codice schermo per '/'
DUEPT = $3A             ; codice schermo per ':'
HZ50 = $80              ; maschera a 50 Hz per TODIN
HZ60 = $7F              ; maschera a 60 Hz per TODIN
AY = $01                ; codice schermo per 'A'
PEE = $10               ; codice schermo per 'P'
EM = $0D                ; codice schermo per 'M'
SCRIVI = 127            ; maschera per rimettere il clock in CR
SCNLOC = $41C           ; indirizzo di schermo dell'orologio
COLLOC = $D81C          ; indirizzo in matrice del video
TRNCLO = $0F            ; maschera per nybble inferiore

* = $C3D0               ; indirizzo iniziale del codice
AMPM *=*+1              ; flag per AM/PM
ORE *=*+1               ; valore ore (per inizializzazione)
MINS *=*+1              ; valore minuti
SECS *=*+1              ; valore secondi
COLOR *=*+1             ; colore orologio
DISPLY *=*+1            ; flag visualizza/nascondi
VETTOR *=*+2            ; deposito per vettore IRQ originale
CONTO *=*+              ; contatore di IRQ
TEMP1 *=*+1

;
; INSERIMENTO DEL "CUNEO"
;

        LDA PALNTS      ; PAL oppure NTSC
        BEQ NTSC        ; diramazione per NTSC
        LDA #HZ50       ; deve essere PAL
        ORA D2CRA
        STA D2CRA       ; fissa TOSIN per 50 Hz
        BMI PALDUN
NTSC
        LDA #HZ60       ; NTSC
        AND D2CRA
        CRA D2CRA       ; fissa per 60 Hz
PALDUN
        LDA #SCRIVI
        AND D2CRB       ; orologio senza sveglia
        STA D2CRB
```

```
          LDA AMPM
          AND #128      ; rende valido il valore di AMPM
          STA AMPM
          LDA ORE       ; estrae le ore
          JRS BINBCD    ; converte in BCD
          ORA AMPM      ; esegue OR con flag di AM/PM
          STA CLOCK+3   ; deposita in CLOCK
          LDA MINS      ; estrae i minuti
          JSR BINBCD    : converte in BCD
          STA CLOCK+2   ; deposita in CLOCK
          LDA SECS      ; estrae i secondi
          JSR BINBCD    ; converte in BCD
          STA CLOCK+1   ; deposita in CLOCK
          LDA #00       ; azzera sempre i 10MI
          STA CLOCK     ; inizia conteggio
;
          SEI           ; disabilita le interrupt
          LDA IRQVET
          STA VETTOR    ; salva vettore IRQ originale
          LDA IRQVET+1
          STA VETTOR+1
;
          LDA #,CUNEO
          STA IRQVET
          LDA #,CUNEO   ; inserisce il cuneo
          STA IRQVET+1
          CLI           ; abilita le interrupt
          RTS

;
; RIMOZIONE DEL CUNEO
;
          SEI           ; disabilita le interrupt
          LDA VETTORQ
          STA IRQVET    ; ripristina il vettore in RAM
          LDA VETTOR+1
          STA IRQVET+1
          CLI           ; abilita le interrupt
          RTS

;
; IL CUNEO INIZIA DA QUI
;
CUNEO
          LDA CONTO
          CMP #RITMO    ; esegue CLOCK a questa IRQ?
          BEQ CONT
ESCI
          JMP FINE      ; no
CONT
          LDA #$FF      ; resetta contatore di IRQ
          STA CONTO
          LDA DISPLY    ; display?
          BEQ ESCI      ; no... dirama

          LDA CLOCK+3   ; estrae ore/AM/PM
          TAX           ; copia nel registro X
          AND #$80      ; estrae AM/PM
          BNE PM        ; diramazione se PM
          LDA #AY       ; visualizza 'A'
          JMP MERIDP
PM
          LDA #PEE      ; visualizza 'P'
MERIDP
          STA SCNLOC+10
          LDA COLOR     ; estrae colore
          STA COLLOC+10; seleziona il colore
          LDA #EM       ; visualizza 'M'
          STA SCNLOC+11
          LDA COLOR     ; seleziona colore
          STA COLLOC+11
;
; CICLO PER LE ORE
;
          TXA           ; estrae le ore
          AND #$10      ; solo la cifra alta
          RSR HICIFRA   ; converte in codice per schermo
          STA SCNLOC    ; lo visualizza
          LDA COLOR
          STA COLLOC    ; seleziona il colore
          TXA           ; riprende il byte
          JSR LODIGT    ; cifra più bassa
          STA SCNLOC+1  ; la visualizza
          LDA COLOR
          STA COLLOC+1  ; seleziona il colore
;
          LDA #DUEPT    ; separazione ore/minuti
          STA SCNLOC+2
          LDA COLOR
          STA COLLOC+2
;
; ADESSO ESEGUE IL CICLO PER I MINUTI
;
          LDA CLOCK+2   ; estrae i minuti
          TAX
          JSR HIDIGT    ; cifra alta
          STA SCNLOC+3  ; la visualizza
          LDA COLOR
          STA COLLOC+3  ; e seleziona il colore
          TXA
          JSR LODIGT    ; cifra bassa
          STA SCNLOC+4  ; la visualizza
;
          LDA #BARRA    ; separatore min/sec
          STA SCNLOC+5
          LDA COLOR
          STA COLLOC+5
;
; ADESSO ESEGUE IL CICLO PER I SECONDI
;
          LDA CLOCK+1   ; estrae i secondi
          TAX
          JSR HIDIGT    ; cifra alta
          STA SCNLOC+6  ; la visualizza
          LDA COLOR
          STA COLLOC+6  ; e seleziona il colore
          TXA           ; riprende il byte
          JSR LODIGT    ; cifra bassa
          STA SCNLOC+7  ; la visualizza
          LDA COLOR
          STA COLLOC+7  ; e seleziona il colore
;
          LDA #PUNTO    ; separatore secondi/decimi
          STA SCNLOC+8
          LDA COLOR
          STA COLLOC+8
;
; ADESSO ESEGUE CICLO PER I DECIMI
;
          LDA CLOCK     ; estrae decimi di secondo
          ADC #CIFRA    ; somma $30 per codice schermo
          STA SCNLOC+9  ; visualizza
          LDA COLOR
          STA COLLOC+9  ; e seleziona il colore
;
;
FINE
          INC CONTO     ; incrementa contatore IRQ
          JMP (VETTOR)  ; esegue normale routine di IRQ

;
; SUBROUTINE
;
HIDIGT
          LSR A
          LSR A         ; sposta nybble alto nel basso
          LSR A
          LSR A
```

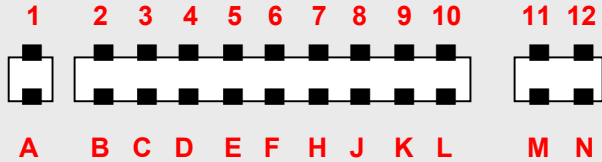
```
          CLC
          ADC #CICFRA    ; somma $30 per codice schermo
          RTS

LODIGT
          AND #TRNCLO    ; rimuove il nybble alto
          CLC
          ADC #CICFRA    ; somma $30 per codice schermo
          RTS

;
; CONVERSIONE DA BINARIO A BCD
;
BINBCD
          LDY #$FF
          SEC
D10
          INY
          SBC #10        ; sottrae fino a valore negativo
          BCS D10
          ADC #10        ; somma 10 al risultato
          STA TEMP1      ; deposita il resto
          TYA            ; numero di 10 sottratti
          ASL A
          ASL A
          ASL A          ; scorrimento nel nybble alto
          ASL A
          ORA TEMP1      ; mette il resto nel nybble basso
          RTS
```

I/O sul Commodore 64

Le porte per le periferiche e le routine del SO

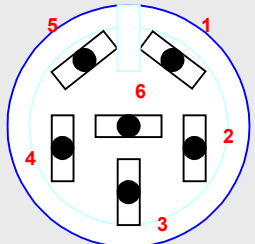
Nella illustrazione qui sotto sono visibili tutte le porte del Commodore 64. Oltre a quella per il registratore, quella per i segnali audio/video e quella per il televisore, ci sono altre tre porte disponibili (da sinistra verso destra): la porta di espansione (usata per le cartucce), la porta seriale (o bus seriale) e la porta utente. Esaminiamole una alla volta.

- *La porta seriale:* a questa presa (una DIN a sei poli) si possono collegare le stampanti seriali Commodore e le unità a dischetti 1541. L'istruzione OPEN si riferisce sempre a questa porta, salvo nel caso in cui si specifichi il numero del dispositivo 2 (che corrisponde ad un'uscita RS232 sulla porta utente). Con il numero di dispositivo 8, ad esempio, il comando OPEN 2,8,2,"NOMEFILE" apre un file sull'unità a dischetti collegata alla porta seriale. Non è consigliare cercare di usare questa porta (né in BASIC né tramite SO) per periferiche diverse da quelle della Commodore.
  In commercio esistono alcune interfacce per convertire i segnali di questa porta da seriali a paralleli IEEE (spesso impiegate per utilizzare, con il C64, periferiche come l'unità a dischetti 4040 progettata per il PET).
- *La porta utente:* il connettore di questa porta, a 24 linee, è del tipo "a pettine". Attraverso di esso possono passare segnali sia paralleli che seriali. Questa porta può essere utilizzata, tra l'altro, per collegare stampanti che esulino dalla gamma Commodore (per esempio una Epson) trattandola come una periferica funzionante tramite RS232. Il modo in cui il SO possa pilotare un dispositivo RS232 è spiegato più avanti.
  La porta utente può anche essere usata per comunicazioni parallele a 8 bit, ma occorre scrivere (o trovare) apposite routine perché non presenti nel SO.
- *Porta di espansione:* il connettore di questa porta, anch'esso a pettine, prevede 44 linee e consente di accedere a tutte le linee di controllo, al bus degli indirizzi ed a quello dei dati del Commodore 64. Questa porta viene utilizzata per interfacciare cartucce di giochi o cartucce con connessione parallela IEEE, adatte al collegamento con periferiche della gamma PET; viene anche utilizzata da dispositivi esterni per acquisire un controllo quasi totale del computer.

Gli schemi di connessione di queste tre porte sono riportati qui sotto.

LA PORTA UTENTE

Note:
(*) Oltre ad essere indirizzabili dalle routine del Kernel, queste linee sono programmabili direttamente dall'utente per operazioni di I/O.
(1) Usata in entrambi i protocolli XON/XOFF e CTS/RTS.
(2) Usata da entrambi i protocolli, ma a livello alto in XON/XOFF.

Piedini superiori		
Piedino	**Sigla**	**Descrizione**
1	GND	Massa
2	—	+5 Volt, 100 mA massimi
3	—	RESET del sistema
4	CNT1	Contatore per la porta seriale CIA #1
5	SP1	Porta seriale CIA #1
6	CNT2	Contatore per la porta seriale CIA #2
7	SP2	Porta seriale CIA #2
8	PC2	Linea di "handshake" dal CIA #2
9	—	Linea collegata alla ATN della porta seriale
10	—	9 VCA fase attiva
11	—	9 VCA fase neutra
12	—	Massa

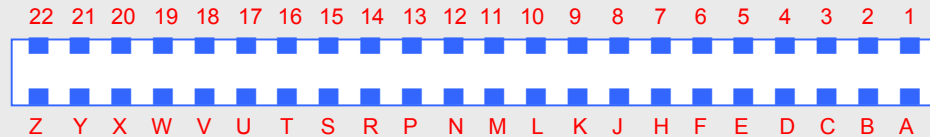
Piedini inferiori				
Piedino	**Sigla**	**Funzione RS232 (*)**		**(Vedere note)**
A	GND	Massa dell'apparecchio		(1)
B	FLAG2	Dato ricevuto	- ingresso	(1)
C	PB0	Dato ricevuto	- ingresso	(1)
D	PB1	Richiesta d'invio (RTS)	- uscita	(2)
E	PB2	Terminale dati pronto (DTR)	- uscita	(2)
F	PB3	Indicatore di anello (RI)	- ingresso	—
H	PB4	Segnale linea ricezione (DCD)	- ingresso	(3)
J	PB5	(Non usato)		—
K	PB6	Abilitazione invio (CTS)	- ingresso	(3)
L	PB7	Gruppo dati pronto (DSR)	- ingresso	(3)
M	PA2	Dati trasmessi	- uscita	(2)
N	GND	Massa dei segnali		(2)

LA PORTA SERIALE

Piedino	Sigla	Descrizione
1	SERIAL SRQIN	Richiesta d'attenzione
2	GND	Massa
3	SERIAL ATN I/O	Segnale di attenzione ai dispositivi
4	SERIAL CLK I/O	Temporizzatore della porta seriale
5	SERIAL DATA I/O	Linea di trasferimento di singoli bit (dati)
6	RESET	Linea di reset dell'hardware

IL KERNEL IN AZIONE

Il Kernel è una raccolta di routine di I/O, ognuna delle quali può essere richiamata da programmi scritti in BASIC o in codice macchina. L'area ROM occupata dal Kernel va da $E000 a $FFF ma le routine vengono chiamate tramite una "tabella di salti" (ossia una sequenza contigua di indirizzi) situata nella parte alta della memoria. Il vantaggio di un accesso tramite tabella di salti è che, indipendentemente dalla versione del Kernel, l'indirizzo di chiamata delle routine non varia, mentre invece può cambiare il codice macchina con il quale vengono scritte.
Per utilizzare le routine del Kernel, servono precise informazioni su ciascuna di esse (in particolare occorre sapere quali parametri sono necessari e quali registri passare); la fonte migliore è sempre il manuale di consultazione per il programmatore e fornito dalla stessa Commodore. Ad esempio: per utilizzare la routine del Kernel per il LOAD dei programmi, occorre un listato come qui descritto. Prima di utilizzarlo, i caratteri che compongono il nome del file, sul quale operare LOAD, vanno depositati, in sequenza, in qualche zona della memoria (qui individuata dall'indirizzo PHI/PLO):

```
LDA   #$01   ; numero logico del file (lfn)
LDX   #$08   ; numero del dispositivo (disco)
LDY   #$00   ; Indirizzo secondario (#$00 attiva la
               LOAD rilocabile)
JSR   $FFBA  ; Chiamata alla routine SLFS del
               Kernel (per fissare lfn e indirizzo
               secondario)
LDA   #$0A   ; Lunghezza del nome del file
               (qui 10)
LDX   PLO    ; Parte bassa dell'indirizzo per il
               nome del file
LDY   PHI    ; Parte alta dell'indirizzo per il nome
               del file
JSR   $FFBD  ; Chiamata alla routine SETNAM del
               Kernel (attribuisci nome del file)
LDA   #$00   ; Load = #$00 / Verify = #$01
LDX   DLO    ; Parte bassa dell'indirizzo di
               destinazione
LDY   DHI    ; Parte alta dell'indirizzo di
               destinazione
JSR   $FFD5  ; Chiamata alla routine LOAD del
               Kernel
```

LA PORTA D'ESPANSIONE

Piedini superiori		
Piedino	**Sigla**	**Commenti**
1	GND	Massa
2	+5V	
3	+5V	
4	$\overline{\text{IRQ}}$	Richiesta d'interrupt
5	R/$\overline{\text{W}}$	Abilitazione lettura/scrittura
6	Dot Clock	
7	I/O 1	
8	$\overline{\text{GAME}}$	
9	$\overline{\text{EXROM}}$	
10	I/O 2	
11	$\overline{\text{ROML}}$	
12	BA	
13	$\overline{\text{DMA}}$	
14	D7	8 linee dati
15	D6	
16	D5	
17	D4	
18	D3	
19	D2	
20	D1	
21	D0	
22	GND	Massa

Piedini inferiori		
Piedino	**Sigla**	**Commenti**
A	GND	Massa
B	$\overline{\text{ROMH}}$	
C	$\overline{\text{RESET}}$	Linea di reset dell'hardware
D	NMI	Interrupt non mascherabile
E	S 02	
F	A15	
H	A14	
J	A13	
K	A12	
L	A11	
M	A10	
N	A9	
P	A8	
R	A7	
S	A6	
T	A5	
U	A4	
V	A3	
W	A2	
X	A1	
Y	A0	
Z	GND	Massa

LA RS-232 SUL COMMODORE 64

Purché si operi con metoro, l'uso della RS232 tramite le routine del sistema operativo non presenta grossi problemi. Per il suo corretto uso, ci sono diverse considerazioni da tener presenti da parte dell'utente (a parte quella di fissare la velocità di trasferimento in baud, esaminata in seguito):

- Il Commodore 64 funziona con livelli di segnale a 0 e +5 volt, mentre lo standard RS232 prevede livelli di –12 e +12 volt. Peranto, salvo nei trasferimenti di dati tra apparecchi Commodore, per rendere tali livelli compatibili occorre un convertitore.
- I codici ASC del Commodore differiscono dai normali codici ASCII, per cui si rende necessario l'impiego di due tabelle di conversione (una per la trasmissione ed una per la ricezione dei dati).
- Ogni volta che viene aperto (OPEN) un canale seriale RS232, il sistema operativo "scarica" (CLR) il contenuto dei registri per cui, automaticamente, va perduto il contenuto di tutte le variabili usate da un programma BASIC e le istruzioni GOSUB generano un messaggio d'errore quando si giunge alla rispettiva RETURN. Questo succede perché le routine del Kernel assegnano, alla RS232, due buffer di 256 byte nella parte alta della memoria, senza tener conto della eventuale presenza di una parte di programma in questa zona, che viene quindi rovinata (per di più tale evento non viene segnalato.
- Se un programma BASIC è lungo oppure contiene molte assegnazioni di stringhe, prima o poi entra in funzione il cosiddetto "garbage collector" (o raccoglitore

di rifiuti) che riorganizza l'utilizzazione della memoria. In tal caso può andar persa parte dei dati ricevuti tramite la RS232. È quindi consigliabile, durante l'utilizzo della porta, usare programmi BASIC di dimensioni limitate e senza troppe assegnazioni di variabili.

Per aprire un canale RS232 dal BASIC si usa il seguente formato:

OPEN 2,2,3,CHR$(CTRL)+CHR$(COM)

dove CTRL e COM sono, rispettivamente, il byte di controllo e di comando che contengono le informazioni necessarie all'apertura del canale. Si noti che CTRL e COM devono essere due valori alfanumerici (oppure le PEEK di due locazioni precedentemente riempite) e non due variabili! Ciascun bit di questi due byte ha un preciso significato che viene riportato nelle tabelle
Per esempio: per aprire un canale RS232 con un bit di stop, una lunghezza dati di 7 bit, a 300 baud (byte CTRL = 38), parità pari, full duplex e flusso su tre linee (byte COM = 96), si utilizza il comando:

OPEN 2,2,3,CHR$(38)+CHR$(96)

Byte CTRL								
Bit								**Funzione**
7	6	5	4	3	3	1	0	
—	—	—	X	0	0	0	1	50 baud
—	—	—	X	0	0	1	0	75 baud
—	—	—	X	0	0	1	1	110 baud
—	—	—	X	0	1	0	0	134,5 baud
—	—	—	X	0	1	0	1	150 baud
—	—	—	X	0	1	1	0	300 baud
—	—	—	X	0	1	1	1	1200 baud
—	—	—	X	1	0	0	0	1800 baud
—	—	—	X	1	0	1	0	2400 baud
—	0	0	X	—	—	—	—	dati 8 bit
—	0	1	X	—	—	—	—	dati 7 bit
—	1	0	X	—	—	—	—	dati 6 bit
—	1	1	X	—	—	—	—	dati 5 bit
0	—	—	X	—	—	—	—	1 bit di stop
1	—	—	X	—	—	—	—	2 bit di stop

X = senza significato

Byte COM								
Bit								**Funzione**
7	6	5	4	3	3	1	0	
—	—	—	—	X	X	X	0	Protocollo a 3 linee
—	—	—	—	X	X	X	1	Protocollo X
—	—	—	0	X	X	X	—	Full duplex
—	—	—	1	X	X	X	—	Half duplex
—	—	0	—	X	X	X	—	Parità ignorata
0	0	1	—	X	X	X	—	Parità dispari
0	1	1	—	X	X	X	—	Parità pari
1	0	1	—	X	X	X	—	Invio Mark (senza parità)
1	1	1	—	X	X	X	—	Invio Spazio (senza parità)

Prima di scegliere la velocità di trasferimento, occorre fare qualche considerazione. In trasmissione la velocità non è particolarmente critica poiché, in genere, gli apparecchi riceventi possono tollerare piccole fluttuazioni (in BASIC si può arrivare a 2400 baud per l'invio attraverso la RS232). Si tenga presente, comunque, che la velocità effettiva di trasferimento è sempre leggermente inferiore ai baud selezionati: in parte a causa della presenza dei bit di stop/parità che allungano la lunghezza totale dei dati.
La situazione, però, è alquanto diversa durante la ricezione dei dati. In questo caso un programma BASIC, all'interno di un ciclo, riesce a malapena a estrarre un byte dal buffer d'ingresso ed a visualizzarlo.
Per migliore l'efficienza della ricezione, occorre, in qualche modo, fermare periodicamente il dispositivo che invia i dati non appena il buffer d'ingresso si riempie. In genere la procedura, nel caso di un protocollo di ricezione a 3 linee, è:

1) Leggere un modesto numero di byte (più sono i baud, minore deve essere questo numero), tramite una GET #2,A$, ed utilizzarli immediatamente oppure depositarli in una matrice in attesa di ulteriore elaborazione.
2) Bloccare temporaneamente il dispositivo esterno di trasmissione usando una PRINT#2,CHR$(17).
3) Leggere i byte rimanenti, fino ad esaurimento, dal buffer d'ingresso della RS232 ed elaborare tutti i byte letti (controllando, nel frattempo, se vengono premuti tasti sulla tastiera se si è raggiunta la fine del file (EOF), ecc.).
4) Riabilitare il dispositivo di trasmissione usando, stavolta, PRINT #2,CHR$(19) e ritornare al punto 1.

Una volta aperto un canale per la RS232, i byte vengono ricevuti o inviati ricorrendo alle semplici PRINT # e GET # (l'istruzione INPUT # deve essere evitata). Il byte dello stato, ST, va periodicamente esaminato per verificare la presenza di errori, segnalati da ST=0 o ST=8. Infine, terminata l'operazione di ingresso/uscita, il canale va chiuso con una CLOSE. Attenzione perché anche questa istruzione, con la RS232, provoca automaticamente lo scarico dei registri.

Linee di comunicazione

L’invio o la ricezione di un blocco di memoria tramite la porta utente

Il Commodore 64 possiede due chip CIA 6526, (Complex Interface Adaptor) dedicati al controllo delle comunicazioni con l’esterno, oltre ad altri chip che gestiscono aspetti specifici del sistema di I/O: il 6510 e quelli per il video. Un chip 6526 prevede due porte a 8 bit per i dati, entrambe dotate di linee programmabili individualmente. Inoltre può gestire comunicazioni a 8 e 16 bit ed incorpora 2 temporizzatori a 16 bit collegabili tra loro. Infine dispone di un registro di scorrimento a 8 bit, per comunicazioni seriali, e di un orologio “giornaliero” programmabile su 24 ore.
Questo chip possiede anche due linee di sincronizzazione con funzioni particolari: PC e Flag. La linea PC viene posta a livello basso (0 logico), per la durata di un ciclo, dopo il trasferimento di un dato sulla porta B del 6526 e può essere usata per segnalare, ad una periferica, la condizione di disponibilità del dato. La linea Flag viene spesso utilizzata come segnale di controllo in ingresso proveniente da un altro apparecchio. Questo segnale può servire per selezionare il bit Flag del registro delle interrupt che, in questa configurazione, provoca la generazione di un interrupt NMI per il processore 6510.
Sul Commodore 64 i due chip 6526 svolgono differenti funzioni di I/O: il CIA #1 (con indirizzo di base $DC00) controlla la tastiera ed i joystick, mentre il CIA #2 (con indirizzo di base $DD00) gestisce i dati sulla porta seriale e su quella per utente. Inoltre il chip per lo schermo controlla l’I/O del monitor ed il 6510 la porta per il registratore a cassette.
Il programma qui presentato illustra, in modo esauriente, le procedure necessarie per programmare direttamente il 6526 per operazioni di ingresso/uscita. La routine è un cuneo NMI che sfrutta la linea Flag. La sua funzione è quella di trasmettere un particolare blocco di memoria, attraverso la porta utente, sotto forma di dati paralleli o, alternativamente, di ricevere dati paralleli a 8 bit finché il blocco di memoria, che li immagazzina, non risulta pieno. Poiché la routine funziona come un cuneo, sincronizza la ricezione o la trasmissione dei dati in base alle interrupt NMI e, contemporaneamente, lascia libero il computer di svolgere altri compiti. Tuttavia, una velocità di trasmissione dei dati troppo elevata, può costringere il Commodore 64 a dedicare tutto il proprio tempo all’esecuzione delle routine di servizio NMI, “congelando” le altre operazioni.
Il programma qui presentato predispone due linee parallele a 8 bit per la comunicazione, tramite la porta utente, con un apparechhio esterno che potrebbe essere, per esempio, un secondo computer o una stampante parallela. I piedini da PB0 a PB7 di questa porta, vengono usati per il trasferimento dei dati. Il piedino Flag 2 è l’ingresso di sincronizzazione del sistema di comunicazione, mentre PA2 e PC2 segnalano, rispettivamente, le condizioni di “pronto per il dato” e “dato valido”. Prima di lanciare il programma è necessario specificare l’area di RAM dalla quale verranno estratti i dati in uscita o ricevuti i dati in ingresso. Gli indirizzi iniziale e finale di quest’area vengono comunicati al programma (in forma byte basso/byte alto) inserenfoli nelle locazioni da 50768 a 50771. La locazione 50772 viene usata per specificare se il programma trasmette o riceve i dati. Nel primo caso la locazione deve contenere il valore 1, nel secondo caso il valore 0. Una volta definiti questi parametri, l’utente può lanciare la routine digitando il comando SYS 50775. Poiché il programma è pilotato da interrupt, le operazioni di trasmissione o ricezione dei dati avvengono “in sottofondo”, ossia contemporaneamente all’introduzione o all’esecuzione di un altro programma BASIC.

PROGRAMMA “CUNEOPAR” DEL COMMODORE 64

Caricatore BASIC

```
1000 REM **CARICATORE BASIC DI CUNEOPAR**
1010 DATA 173,84,198,208,61,169,0,141,3
1020 DATA 221,169,144,141,13,221,173,2
1030 DATA 221,9,4,141,2,221,173,0,221,9
1040 DATA 4,141,0,221,173,80,198,133,251
1050 DATA 173,81,198,133,252,173,24,3
1060 DATA 141,85,198,173,25,3,141,86,198
1070 DATA 120,169,188,141,24,3,169,198
1080 DATA 141,25,3,88,96,169,255,141,3
1090 DATA 221,169,144,141,13,221,173,57
1100 DATA 3,141,85,198,173,25,3,141,86
1110 DATA 198,120,169,234,141,24,3,169
1120 DATA 198,141,25,3,88,96,169,144,44
1130 DATA 13,221,240,36,173,1,221,145
1140 DATA 251,230,251,208,2,230,252,173
1150 DATA 82,198,197,251,173,83,198,229
1160 DATA 252,144,49,173,0,221,41,252
1170 DATA 141,0,221,9,4,141,0,221,108,85
1180 DATA 198,169,144,44,13,221,240,246
1190 DATA 177,251,141,1,221,230,251,208
1200 DATA 2,230,252,173,82,198,197,251
1210 DATA 173,83,198,229,252,144,3,108
1220 DATA 85,198,120,173,85,198,141,24,3
1230 DATA 173,86,198,141,25,3,88,108,24
1240 DATA 3
125 DATA 25596: REM **CHECSUM**
1260 CC=0
1270 FOR I=50775 TO 50971
1280 READ X: CC=CC+X: POKEI,X
1290 NEXT
1300 READ X: IF CC<>X THEN PRINT “CHECSUM
      ERROR”
1310 END
```

Caricamento di “Cuneopar”

Il listato in Assembly del programma “Cuneopar(allelo)” va caricato in memoria e tradotto in codice macchina con un assembler. In alternativa il programma può essere trascritto sotto forma di istruzioni DATA (vedi pagina precedente), per poi lanciare il programma di caricamento in BASIC.

Listato Assembly

```
;*****************************************************************
;*                                                               *
;*   CUNEOPAR – PROGRAMMA DI TRASMISSIONE                        *
;*         e RICEZIONE PER COMUNICAZIONI                         *
;*      PARALLELE A 8 BIT SUL COMMODORE 64                       *
;*                                                               *
;*****************************************************************
;
          CIA2 = $0000          ; indirizzo di base del chip 6526
          USCITA = $FF
          INGRESSO = $00
          OUTSHK = $04
          INTSHK    = $90
          TOGHI = $04
          TOGLO = $FC
          NMIVETT = $0318
          ZPTEMP = $FB
          * = $C650
          INIZIO *=*+2          ; indirizzo iniziale
          FINE *=*+2            ; indirizzo finale
          MODO *=*+1            ; flag di ingresso/uscita
          VETTORE *=*+2         ; memoria per il vettore NMI

          LDA MODO              ; ingresso o uscita
          BNE OUTDAT            ; salta se ingresso
          LDA #INGRESSO
          STA CIA2+3            ; seleziona il DDR (reg. direz. dati)
                                  per l’ingresso
          LDA INTMSK
          STA CIA2+13           ; disabilita le interrupt di Flag
          LDA CIA2+3
          ORA #OUTSHK
          STA CIA2+2            ; seleziona PA2 per l’uscita
          LDA CIA2
          ORA TOGHI
          STA CIA2              ; pone alta la linea di
                                  “handshake” PA2
          LDA INIZIO
          STA ZPTEMP
          LDA INIZIO+1          ; sposta puntatori in pagina 0
          STA ZPTEMP+1
;
; INIZIALIZZA IL CUNEO D’INGRESSO
;
          LDA NMIVETT
          STA VETTORE           ; salva il vecchio vettore NMI
          LDA NMIVETT+1
          STA VETTORE+1
          SEI                   ;
          LDA #<NXTIN           ;
          STA NMIVETT           ; inserisce il cuneo di ingresso-dati
          LDA #>NXTIN           ;
          STA NMIVETT+1         ;
          CLI
          RTS
;
; INIZIALIZZA IL CUNEO D’USCITA
;
OUTDAT
          LDA #USCITA
          STA CIA2+3            ; seleziona il DDR per l’uscita
          LDA #INTMSK
          STA CIA2+13           ; disabilita le interrupt di Flag
;
;
          LDA NMIVETT
          STA VETTORE
          LDA NMIVETT+1         ; salva il vecchio vettore NMI
          STA VETTORE+1
          SEI                   ;
          LDA #<NXTOUT          ;
          STA NMIVETT           ; inserisce il cuneo di uscita-dati
          LDA #>NXTOUT          ;
          STA NMIVETT+1
          CLI                   ;
          RTS
;
;
; ROUTINE DI SERVIZIO PER L’INGRESSO DI DATI
;
;
NXTIN
          LDA #INTMSK           ; verifica ICR
          BIT CIA2+13           ; interrupt causata da Flag?
          BEQ NOTCOM            ; no... una comune NMI
;
; BYTE SULLA PORTA
;
          LDA CIA2+1            ; legge un byte
          STA (ZPTEMP),Y        ; lo deposita in memoria
          INC ZPTEMP
          BNE TEST1             ; incrementa il puntatore
          INC ZPTEMP+1
;
TEST1
          LDA END
          CMP ZPTEMP
          LDA END+1             ; verifica se finito
          SBC ZPTEMP+1
          BCC DONE              ; se finito, salta
;
; COMUNICA DI ESSERE PRONTO PER BYTE SUCCESSIVO
;
          LDA CIA2
          AND #TOGLO
          STA CIA2              ; pone PA2 prima bassa, poi alta
          ORA #TOGHI
          STA CIA2
;
; ESEGUE LA NORMALE ROUTINE NMI
;

NOCOM
          JMP (VETTORE)


;
; ROUTINE DI SERVIZIO PER L’USCITA DI DATI
;
;
NXTOUT
          LDA #INTMSK           ; verifica ICR
          BIT CIA2+13           ; interrupt causata dal Flag?
          BEQ NOTCOM            ; no... una comune NMI
;
; BYTE TRASMESSO
;
          LDA (ZPTEMP),Y        ; estrae un byte dalla memoria
          STA CIA2+1            ; lo trasmette. (PC passa a livello basso
                                  per 1 ciclo)
          INC ZPTEMP
          BNE TEST2             ; incrementa il puntatore
          INC ZPTEMP+1

TEST2
          LDA END
          CMP ZPTEMP
          LDA END+1             ; verifica se finito
          SBC ZPTEMP+1
          BCC DONE              ; se finito, salta
;
; ESEGUE LA NORMALE ROUTINE NMI
;
          JMP (VETTORE)
;
;
; FINITA RIMOZIONE CUNEO
;
;
DONE
          SEI
          LDA VETTORE
          STA NMIVETT           ; ripristina il valore originale
          LDA VETTORE+1         ; nel vettore NMI
          STA NMIVETT+1
          CLI
          JMP (NMIVET)
```

Effetti ottici

Il funzionamento del chip del Commodore 64

Il Commodore 64 possiede 8 modi “base” per la grafica dello schermo. Nella grafica in bassa risoluzione, il set di caratteri può risiedere in ROM o in RAM ed essere visualizzato in uno dei tre seguenti modi: Standard, Multicolour ed Extended colour; in alta risoluzione i modi sono due: Standard e Multicolour. Oltre a questi modi base, il Commodeore 64 consente variazioni sul 38 colonne, anziché sulle normali 40, e su 24 righe invece delle normali 25. Di solito tali variazioni vengono usate con la funzione di scroll omogeneo, orizzontale e verticale, dello schermo. Sul Commodore 64 questa funzione produce risultati migliori se si ricorre al codice macchina in quanto la progressiva sostituzione dell’immagine deve avvenire, con grande rapidità, all’interno della RAM del video.
Se in un programma di notevoli dimensioni viene usato lo schermo ad alta definizione, la memoria diventa un bene prezioso. Tuttavia, nell’indirizzamento degli schermi ad alta e bassa risoluzione, il Commodore 64 consente una notevole libertà di movimento. Come si vedrà in seguito, se il tracciamento viene eseguito in codice macchina, è possibile collocare uno schermo ad alta risoluzione nella RAM “nascosta” dalla ROM dell’interprete BASIC. L’utilità di una simile soluzione è ovvia: se il programma è esclusivamente in codice macchina, l’interprete ROM occupa inutilmente ben 8 Kbyte di preziosa memoria. Per una coincidenza, 8 Kbyte è proprio l’area di memoria necessaria per uno schermo ad alta risoluzione.
Il chip d’interfaccia per lo schermo 6566/67 (più brevemente VIC-II), ha il compito di generare i dati per l’immagine video che vengono poi inviati al televisore o al monitor. Per far ciò, il VIC-II deve essere in grado di leggere tali dati dalla RAM o dalla ROM. Vale la pena di analizzare come il VIC-II ottenga le proprie informazioni ricordando che questo è uno dei processi meno evidenti del Commodore 64.

Il carattere ‘A’ nel modo di schermo “normale”							
0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0

Il carattere ‘A’ nel modo di schermo Multicolour			
0 0	0 1	1 0	0 0
0 0	1 1	1 1	0 0
0 1	1 0	0 1	1 0
0 1	1 1	1 1	1 0
0 1	1 0	0 1	1 0
0 1	1 0	0 1	1 0
0 1	1 0	0 1	1 0
0 0	0 0	0 0	0 0

Colori coordinati
Nel modo di schermo “normale” del Commodore 64, ogni bit posto a 1, all’interno degli 8 byte che definiscono un carattere, viene visualizzato nel colore di primo piano attuale. I bit posti a 0 vengono invece visualizzati nel colore di sfondo dello schermo.
Quando viene selezionato il modo Multicolour, i bit non vengono più interpretati singolarmente, ma a coppia. Le quattro combinazioni possibili, di una coppia di bit, rappresentano il colore di primo piano, quello di sfondo ed i due “multicolour” supplementari.

Modo Multicolour

Lo schermo in bassa risoluzione ed il modo “bit map” sono descritti a pag 694 e 817 (IL MIO COMPUTER), pertanto qui vengono esaminati due degli altri modi previsti per la grafica su Commodore 64.
Il modo Multicolour consente di visualizzare 4 colori diversi, anziché solo 2, in un’unica cella di carattere.
L’unico inconveniente è che la risoluzione orizzontale diventa a coppie di pixel e non a pixel singoli. Il modo grafico Multicolour può essere usato sia in alta che in bassa risoluzione ma, nel primo caso, la determinazione dei colori dei punti è un po’ diversa. Queste istruzioni BASIC abilitano e disabilitano il modo Multicolour:

```
POKE 53270, PEEK(53270) OR 16
POKE 53270, PEEK(53270) AND 239
```

Se il modo Multicolour viene usato in bassa risoluzione e se il bit 4 del “nybble” (4 bit o semibyte) del colore associato vale 1, allora il carattere viene interpretato secondo il modo Multicolour ed i tre bit più bassi ne definiscono il colore. Ciò significa che i caratteri con “nybble” di colore associato, compresi tra 0 e 7, vengono visualizzati normalmente, mentre se il codice di colore è compreso tra 8 e 15, il carattere viene visualizzato nel modo Multicolour. I colori delle coppie di pixel vengono definiti secondo lo schema della tabella:

Config. binaria della coppia di pixel	Colore	Determinato da
0 0	Sfondo dello schermo	53281 ($D021) Bit 0-4
0 1	Multicolour #1	53282 ($D022) Bit 0-4
1 0	Multicolour #2	53283 ($D023) Bit 0-4
1 1	Colore di primo piano	Bit 0-3 del “nybble” di colore

Agendo sul contenuto degli indirizzi 53282 e 53283 è possibile cambiare istantaneamente il colore di qualsiasi coppia di pixel Multicolour associati. È da notare che il modo Multicolour produce risultati migliori se usato con caratteri defini dall’utente poiché le configurazioni di bit traggono vantaggio dal fatto di essere interpretate come coppie di pixel.
Un altro modo grafico disponibile sul Commodore 64 è il modo Extended colour (a colori estesi) che permette di controllare il colore di sfondo nei primi 64 caratteri del set. Questo modo, che non può venire usato insieme al Multicolour, viene abilitato e disabilitato dalle seguenti linee BASIC:

```
POKE 53265, PEEK(53265) OR 64
POKE 53265, PEEK(53265) AND 191
```

Lo stesso carattere può apparire sullo schermo con un massimo di 4 diversi colori di sfondo, uno dei quali è il colore di sfondo dello schermo. I caratteri, successivi al sessantaquattresimo, non possono essere visualizzati nel modo Extended perché i bit 6 e 7, del codice di schermo, vengono usati per controllare indirettamente il colore del carattere. Il codice di schermo per la ‘A’, per esempio, è 1 mentre quello di una ‘A’ in campo inverso è 65. Tuttavia un’istruzione POKE 65, se lo schermo è selezionato nel modo Extended colour, non produce una ‘A’ in campo inverso, bensì una ‘A’ normale con un colore di sfondo definito dal contenuto dell’indirizzo 53282 ($D022). Anche con l’istruzione POKE 129 si ottiene una ‘A? normale ma con un colore di sfondo definito dal contenuto dell’indirizzo 53283 ($D023). Ta tabella sotto mostra la corrispondenza tra i codici di schermo ed i registri del colore.

Codice di schermo	Bit 7	Bit 6	Registro colore di sfondo
64 - 127	0	1	53282 ($D022) Extended colour 1
128 - 191	1	0	53283 ($D023) Extended colour 2
192 - 255	1	1	53284 ($D024) Extended colour 3

Gestione della memoria

Il VIC-II “vede” in modo diverso, e molto più semplice, la mappa di memoria usata dal 6510. Di volta in volta il VIC-II può accedere solo ad uno dei 4 banchi di memoria da 16 Kbyte. Il VIC-II esamina la memoria, quindi, come attraverso i vetri di una “finestra” le cui dimensioni sono un blocco di 16 Kbyte. L’indirizzo di base di questa finestra è controllato dal software e può assumere quattro valori diversi.

```
1000 REM ** SELEZIONA IL BANCO PER LA FINESTRA DEL VIC **
1010 POKE 56578, PEEK(56578) OR 3: REM DDR CIA #2 BIT 0,1 PER L’OUTPUT
1020 FIN=3: REM SELEZIONA LA FINESTRA NORMALE
1030 POKE 56576, (PEEK(56576) AND 252) OR FIN): REM PORTA A CIA #1 BIT 0,1
```

In questo caso FIN può assumere valori compresi tra 0 e 3. In qualsiasi momento l’istruzione PEEK(56576) AND 3 permette di conoscere il valore corrente di FIN. L’indirizzo in memoria, corrispondente all’inizio della finestra a 16 Kbyte, può essere calcolato con la formula:

VFIN = 16384*(3 – FIN)

Ai valori registrati in FIN corrispondono i seguenti indirizzi:

FIN	Indirizzo iniziale della finestra
0	49152 ($C000)
1	32768 ($8000)
2	16384 ($4000)
3	0 ($0000)

All’interno della finestra del VIC-II, il 6510 si aspetta di trovare la memoria video ed un’immagine formata da caratteri ROM in bassa risoluzione (o valori numerici per l’alta risoluzione se questa è stata selezionata), come pure le informazioni relative agli sprite. I puntatori, a ciascuno degli otto sprite, si trovano alla fine della memoria video (quindi devon essere spostati assieme allo schermo).

Lo scarto (offset) dell'inizio della memoria video, rispetto alla base dell'attuale finestra del VIC-II, è controllato dai quattro bit superiori del registro di controllo del VIC-II situato all'indirizzo 53272 ($D018). Usando questi bit si può collocare lo schermo in uno dei blocchi a 16 Kbyte all'interno della finestra:

```
1040 REM ** SELEZIONA L'OFFSET TRA LO SCHERMO E LA BASE DELLA FINESTRA **
1050 SO=1: REM SELEZIONA L'OFFSET NORMALE
1060 POKE 53272, (PEEK(53272) AND 15) OR 16*SO)
```

In questo caso SO può assumere valori compresi tra 0 e 15 e l'istruzione PEEK(53272) AND 15 permette di sapere, in qualsiasi momento, tale valore.
L'indirizzo in memoria, corrispondente alla base dello schermo corrente, può essere calcolato con la formula

SC = VFIN+1024*SO

(base della finestra + offset), oppure con:

SC = 16384*(3 - (PEEK(56576) AND 3)) + 64 – (PEEK(53272) AND 240)

Tuttavia c'è un piccolo problema relativo al movimento dello schermo: la RAM del colore non può essere spostata. Quindi, per avere uno schermo "alternativo", è necessario ricorrere ad una piccola routine in codice macchina capace di trasferire e richiamare la memoria del colore dda un "buffer". Questa routine è già stata descritta a pag. 1398 (IL MIO COMPUTER) per il programma degli schermi aggiuntivi sul Commodore 64.
Un secondo fattore da considerare è che, se si desidera visualizzare sul nuovo schermo, è necessario comunicare al sistema operativo (piuttosto che al chip video) la sua posizione in memoria utilizzando un'istruzione POKE (oppure STA) per indirizzare 648 ($0288), ossia il puntatore di base della "memoria video". Se SC viene calcolato nel modo descritto in precedenza, occorre usare la seguente istruzione BASIC di indirizzamento:

```
POKE 648, INT(SC/256)
```

Per selezionare l'indirizzo di base del set dei caratteri o dello schermo in alta risoluzione, si usano le linee:

```
1070 REM ** SELEZIONA L'OFFSET DALLA BASE DELLA FINESTRA DI MC/HIRES **
1080 HO=4: REM SELEZIONA L'OFFSET NORMALE
1090 POKE 53272,(PEEK(53272) AND 240) OR 2*HO)
```

In teoria HO può assumere qualsiasi valore compreso tra 0 e 7 ma, in pratica, esistono alcuni fattori che limitano la scelta. Per FIN uguale a 1 o 3, HO non può avere il valore di 2 o 3 dato che ciò si verifica quando il VIC-II "vede" l'immagine normale dei caratteri ROM. Inoltre valori elevati di HO pongono la parte superiore della memoria, di alta risoluzione, fuori dalla portata del VIC-II. L'indirizzo in memoria, corrispondente alla base dello schermo corrente per la matrice di caratteri e per l'alta risoluzione, viene calcolato come:

MC 0 VFIN + 2048 * HO
(base della finestra + offset), oppure con l'espressione:

MC = 16384 * (3 – (PEEK(56576) AND 3)) + 1024 * (PEEK(53272) AND 14)

Selezionando tali registri è possibile spostare a piacimento l'intera memoria dell'immagine video.

Schermi multipli

Testi e grafica in alta risoluzione simultaneamente sullo schermo

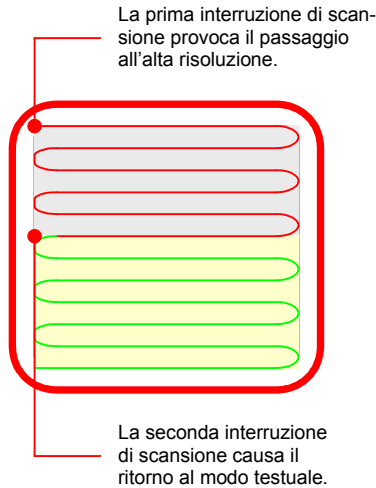
Per visualizzare caratteri e simboli, il chip VIC-II deve leggerli dalla memoria e ciò avviene facendogli accedere a parte delle linee del bus indirizzi ed a tutte le linee del bus dati. La procedura di lettura della RAM e della ROM, da parte del VIC-II in combinazione con il proceddore 6510, viene chiamata “accesso diretto alla memoria” (DMA, Direct Memory Access). Idealmente il VIC-II usa le linee degli indirizzi e dei dati nei momenti in cui queste non sono utilizzate dal processore, per cui le operazioni dei due chip avvengono in modo reciprocamente “trasparente”, cioè non interferiscono tra di loro. Sfortunatamente, però, il 6510 è un processore abbastanza semplice: è dotato di pochi registri interni anche se nessuna istruzione impiega più di sette cicli macchina (in media tre o quattro cicli).
Poiché il VIC-II deve leggere grandi quantità di dati, specie quando si tratta di visualizzare più sprite, non ha il tempo materiale per operare in modo trasparente; per questo motivo il chip è stato dotato di una speciale linea, chiamata BA (Bus Available, “bus disponibile”) per segnalare al 6510 che il bus è occupato. Solo così il VIC-II può espletare le proprie funzioni di visualizzazione. Quando la linea BA è posta a livello logico basso, il 6510 viene informato che il bus è richiesto dal VIC-II. Prima che venga emesso un altro segnale che abilita l’accesso del 6510 al bus, al processore viene dato il tempo necessario a completare l’istruzione in corso. Questo segnale è chiamato “Address Enable Control” (controllo di abilitazione del bus) o AEC.
Considerate le unità di misura in gioco, la quantità di tempo “rubata” al 6510 è piuttosto grande: si pensi che gli indirizzi dei puntatori ai caratteri devono esser letti ogni volta che sullo schermo viene effettuata la scansione di otto linee (i caratteri sono formati da otto righe di pixel) e che ogni linea impica 40 accessi consecutivi per estrarre gli indirizzi dei puntatori alla memoria video (ci sono 40 caratteri su ogni riga di schermo). Nel sistema PAL il chip del video aggiorna ciascuna delle 625 linee dello schermo (procedendo a linee alterne) circa 25 volte al secondo. Il sistema americano NTSC prevede solo 524 linee ma con una frequenza di aggiornamento di 30 volte al secondo. Tutto ciò richiede del tempo e, con un calcolo approssimato, si può affermare che il 6510 viene rallentato di un buon 15 o 20 per cento.
Queste momentanee “assenze” del 6510 non comportano alcun inconveniente, tranne quando si debba operare in tempo reale. È il caso, per esempio, delle operazioni di lettura o scrittura su nastro in cui è fondamentale la sincronizzazione. Ecco perché si rende necessario bloccare la visualizzazione su schermo (con POKE 53265,11) prima dell’operazione di ingresso/uscita, per riabilitarla in seguito (con POKE 53265,27). La necessità di sospendere la visualizzazione dipende dal modo di funzionare della periferica con la quale il computer intende comunicare: nelle operazioni sui dischetti, ad esempio, non occorre disabilitare lo schermo.

Suddividere lo schermo

Sul Commodore 64, con qualche accorgimento di programmazione, è possibile suddividere lo schermo in modo da utilizzare contemporaneamente la grafica sia in alta che in bassa risoluzione. Come spesso accade nei giochi in commercio, la metà superiore dello schermo raffigura uno scenario in grafica ad alta risoluzione mentre, nella parte inferiore, vengono visualizzati i messaggi e l’input del giocatore. Normalmente non è possibile mescolare i due tipi di grafica.
Il programma *Spliscreen*, presentato più avanti e che illustra molti dei concetti finqui esposti, opera in modo abbastanza semplice: nella RAM “nascosta” sotto la ROM del BASIC, viene creato uno schermo in alta risoluzione che viene ininterrottamente visualizzato nel terzo superiore dello schermo. Allo stesso tempo i due terzi inferiori rimangono nella grafica a bassa risoluzione. Nel primo paragrafo (**Il SO del Commodore 64**) abbiamo visto come utilizzare i vettori RAM modificandone l’indirizzo affinché vengano eseguite routine in codice macchina scritte dall’utente. Per suddividere lo schermo, qui viene operato un intervento, piuttosto insolito, sul vettore IRQ. La routine abbinata a questo vettore viene chiamata in diverse occasioni, una delle quali si presenta quando un temporizzatore di uno dei CIA modifica un bit nel registro segnalatore delle interrupt (IFR, Interrupt Flag Register) che ha per indirizzo 53273 ($D019). Ciò avviene circa ogni sessantesimo di secondo, provocando l’esecuzione della routine di servizio IRQ. Tra le altre cose, questa routine effettua una scansione della tastiera per vedere se si sta premendo un tasto. Alcuni bit del registro IFR vengono posti a 1 anche a seguito di certi eventi controllati dal chip VIC-II; per esempio quando avviene una collisione tra due sprite o quando il contatore di scansione raggiunge un valore prefissato. Esiste un altro registro che influenza la linea IRQ del 6510: il registro di abilitazione delle interrupt (IER, interrupt Enable Register) ed il cui indirizzo è 53274 ($D01A). Se il programmatore modifica i bit di IER, allora i corrispondenti bit di IFR provocano una chiamata alla routine di IRQ.

Ecco i principi del programma *Splitscreen:*

1) Quando si verifica l'interruzione nella scansione in cima allo schermo, il programma seleziona il modo "mappa di bit", predispone un'interruzione di scansione a circa metà dello schermo, e poi esegue un ritorno da interrupt (RTI).

2) Quando si verifica l'interruzione di scansione a metà schermo, il programma riseleziona la bassa risoluzione e predispone un'interruzione di scansione all'inizio dello schermo. Di nuovo esegue un ritorno da interrupt (RTI).

Si noti la curiosa configurazione di questo sistema. Parte della normale memoria video (corrispondente al terzo superiore dello schermo) viene utilizzata per conservare informazioni sul colore per la grafica in alta risoluzione, mentre il resto dello schermo viene utilizzato normalmente in bassa risoluzione. Nonostante tutto, rimane da superare un inconveniente. Le interruzioni di scansione si verificano circa ogni cinquantesimo di secondo e devono essere "servite" immediatamente altrimenti, nel frattempo, il raggio di scansione prosegue nel suo spostamento ed i dati verrebbero visualizzati in posizione errata. Di solito la routine di IRQ viene innescata ogni sessantesimo di secondo e la sua esecuzione ha una durata relativamente lunga. Se questa inizia subito prima di un'interruzione di scansione, c'è il rischio che la routine di "cuneo", inserita dal nostro programma, venga eseguita con troppo ritardo provocando un fastidioso sfarfallio sul margine di separazione dei due schermi.

La soluzione è abbastanza semplice: si tratta di forzare l'esecuzione della routine di IRQ subito dopo quella della routine di cuneo relativa alla scansione. Ciò si ottiene spegnendo il normale temporizzatore per i sessantesimi di secondo (disabilitando così il normale innesco della routine IRQ) e saltando alla routine IRQ subito dopo aver eseguito il cuneo. Con questo accorgimento si garantisce la necessaria sincronizzazione anche se implica una minor frequenza di scansione della tastiera che, del resto, non ha alcuna conseguenza pratica.

Infine rimane da collocare lo schermo in alta risoluzione nella RAM normalmente "nascosta" dalla ROM del BASIC. L'unico modo per accedere convenientemente a questa RAM, adoperando il BASIC, è ricorrere a delle POKE ma, se il programma è in codice macchina, l'intera ROM del BASIC non serve e può essere disabilitata liberando, così, 8 Kbyte di memoria. Comunque, con qualche modifica al listato qui riportato, è possibile spostare altrove lo schermo in alta risoluzione per potervi accedere anche in ambiente BASIC.

PROGRAMMA SPLITSCREEN PER COMMODORE 64

Caricatore in BASIC

```
 900 REM **************************************************
 902 REM **      CARICATORE BASIC PER              **
 903 REM **            SPLITSCREEN 64              **
 910 REM **   NOTA:  IN QUESTA VERSIONE            **
 920 REM **      LA HIRES E' SOTTO BASIC           **
 930 REM **    E NON SI PUO' USARE PEEK            **
 940 REM **   NELLA SUBROUTINE DI  PLOT            **
 950 REM **************************************************
 960 :
1000 REM CARICAMENTO SPLITSCREEN E PROVA **
1010 DATA 76,50,192,49,234,173,25,208
1020 DATA 240,37,169,255,160,21,141,25
1030 DATA 208,173,0,221,73,2,141,0,221
1040 DATA 173,17,208,73,32,141,17,208,41
1050 DATA 32,240,4,169,121,160,255,141
1060 DATA 18,208,140,24,208,108,3,192
1070 DATA 173,14,220,41,254,141,14,220
1080 DATA 173,20,3,141,3,192,173,21,3
1090 DATA 141,4,192,169,5,141,20,3,169
1100 DATA 192,141,21,3,173,17,208,41,95
1110 DATA 141,17,208,169,0,141,18,208
1120 DATA 169,255,141,25,208,169,1,141
1130 DATA 26,208,165,1,41,254,133,1,169
1140 DATA 0,133,247,169,160,133,248,160
1150 DATA 0,162,28,169,0,145,247,200,208
1160 DATA 251,230,248,202,208,246,169
1170 DATA 188,133,248,160,0,162,4,169
1180 DATA 183,145,247,200,208,251,230
1190 DATA 248,202,208,246,165,1,9,1,133
1200 DATA 1,96,255
1210 DATA 20633: REM * CHECSUM *
1220 CC=0: FOR I=0 TO 160
1230 READ x: POKE 49152+I,X: REM INSER. CODICE
1240 CC=CC+X: NEXT
1250 READ X: IL X <> CC THE PRINT "ERRORE": STOP
1260 REM ** VERIFICA SUDDIVISIONE SCHERMO **
1270 SYS 49152: REM CHIAMATA A SPLITSCREEN
1280 CM=40960: REM INIZIO HIRES
1290 REM ** TRACCIA ASSI VERTICALI **
1300 X=160: FOR Y=0 TO 70
1310 GOSUB 1400: REM CALCOLA INDIRIZZO
1320 POKE M+R, 2^(7-C): REM TRACCIA PUNTO
1330 NEXT
1340 REM ** TRACCIA ASSI ORIZZONTALI **
1350 Y=37: FOR X=0 TO 319
1360 GOSUB 1400: REM CALCOLA INDIRIZZO
1370 POKE M+R,255: REM TRACCIA SEGMENTO
1380 NEXT
1390 END
1400 REM **S/R DI CALCOLO**
1410 U=INT(Y/8): V=INT(X/8)
1420 R=Y AND 7: C=X AND 7
1430 M=CM+(40*U+V)*8
1440 RETURN
```

Listato in Assembly

```
;++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++
;++                        ++
;++       SPLITSCREEN      ++
;++                        ++
;++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++

        ;
        * = $C000
        JMP INIZIO
        ;
        MEM1 = $F7
        MEM2 = $F9
        MEM3 = $FB
        SCN = $FD
        CINV = $314     ; vettore IRQ
        TEMPIRQ *=*+2
        ;
NEWIRQ = *              ; nuovo inizio cuneo IRQ
        ;
        LDA $D019       ; esamina stato interrupt
        BEQ NOTVIC      ; IRQ proviene da VIC?
        LDA #$FF
        LDY #$15
        STA $D019       ; azzera memoria interrupt
        LDA $DD00
        EOR #$02
        STA $DD00       ; scambio dei banchi di mem.
        LDA $D011
        EOR #$20        ; cambia il bit del Bit Map Mode
        STA $D011
        AND #$20        ; esamina il bit del BMM
        BEQ LSCN        ; parte inferiore dello schermo
        LDA #$79        ; fissa scansione alla linea 22
        LDY #$FF
LSCN
        STA $D012       ; deposita n. val. di scansione
        STY $D018
NOTVIC
        JMP (TEMPIRQ)
        ;
INIZIO =*               ; inizializza il cuneo IRQ
        ;
        LDA $DC0E
        AND #$FE        ; disabilita il timer A
        STA $DC0E       ; CIA #1
        LDA CINV
        STA TEMPIRQ     ;
        LDA CINV+1      ; cambia il vettore IRQ
        STA TEMPIRQ+1;
        LDA #<NEWIRQ
        STA CINV
        LDA #>NEWIRQ
        STA CINV+1
        LDA $D011
        AND #$5F        ; reset del Bit Mode
        STA $D011
        LDA #$00
        STA $D012
        LDA #$FF
        STA $D019       ; reset memoria IRQ
        LDA #$01
        STA $D01A       ; abilita confronto scansione
        LDA $01
        AND #$FE
        STA $01         ; rimuove il banco ROM del BASIC
        LDA #$00
        STA MEM1        ; prepara i puntatori in pagina 0
        LDA #$A0        ; per area hires “dietro” la ROM
        STA MEM1+1      ; del BASIC
        LDY #$00
        LDX #$1C
        LDA #$00        : hires
DDD
        STA (MEM1),Y
        INY
        BNE DDD
        INC MEM1+1
        DEX
        BNE DDD
        LDA #$BC
        STA MEM1+1
        LDY #$00
        LDX #$04
        LDA #$B7        ; fissa colore
DDE
        STA (MEM1),Y
        INY
        BNE DDE
        INC MEM1+1
        DEX
        BNE DDE
        LDA $01
        ORA #$01
        STA $01         ; ripristina ROM del BASIC
        RTS             ; ritorna al BASIC
```

Il “beat” dei byte

L’oscillatore

L’oscillatore è un circuito elettronico che produce dei segnali ripetitivi. Quando questi segnali vengono amplificati e passati ad un altoprlante, si ottengono dei suoni o delle note di toni diversi. Negli home computer ci sono fino a quattro di questi oscillatori: ovviamente più ce ne sono, più note si possono generare contemporaneamente.
Tre sono i parametri che descrivono i suoni generati: la frequenza, l’inviluppo (in cui è compreso anche il volume) e la forma d’onda.

La frequenza

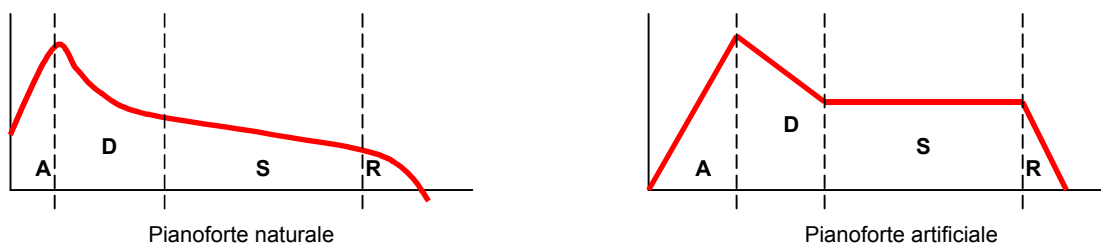
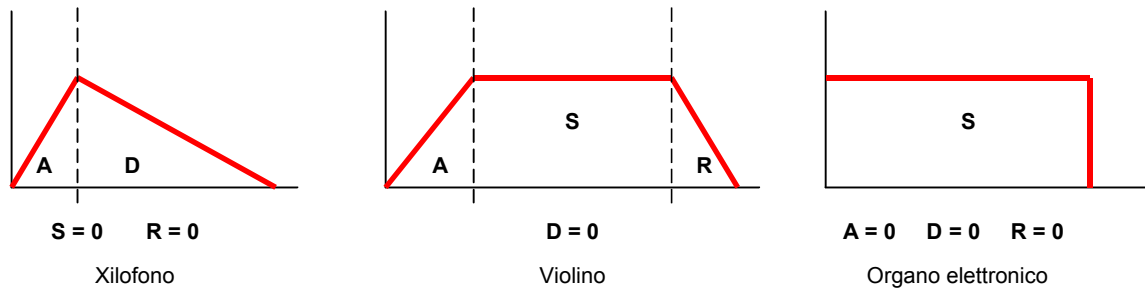
Questa è la caratteristica più importante da controllare poiché è proprio la frequenza che determina la nota generata. Il termine frequenza indica il numero di volte in cui si ripete un segnale nell’intervallo di un secondo e viene misurata in Hetz (Hz) o cicli al secondo. I suoni percepiti dall’orecchio umano hanno una frequenza compresa, grosso modo, tra 20 e 10.000 Hz (10 KHz). Anche se le frequenze al di sotto di 20 Hz non sono percepibili, le possiamo ugualmente utilizzare per modificare le caratteristiche di un suono percepibile. Tale tecnica si chiama “modulazione” e, attualmente, è possibile applicarla solamente al Commodore 64 (ovviamente nella fascia degli hom computer).
Detto questo, tuttavia non occorre una conoscenza approfondita delle varie frequenze; basta sapere come si ottengono le note musicali. Con alcuni computer il compito è molto facile: esistono comandi BASIC che calcolano automaticamente la frequenza di ciascun suono e nin rimane altro che indicare l’intensità del suono stesso oppure la nota voluta ma, rigorosamente, secondo la notazione anglosassone (in lettere): A, A# B, ecc. (corrispondenti a LA LA#, SI, ecc.). Purtroppo sul Commodore 64 il sistema è più complesso: occorre utilizzare una tabella di conversione, nella quale sono indicate le frequenze corrispondenti alla nota voluta, insieme ai valori che devono essere depositati (con l’impiego dell’istruzione POKE), in una particolare locazione di memoria, La tabella riportata qui sotto illustra la conversione delle note corrispondenti alla scala di “do centrale”; questa dovrebbe risultare utile a chi volesse cimentarsi nella generazione di musica in linguaggio macchina.

Dalla nota musicale alla frequenza
Per calcolare la frequenza di ciascun suono della scala musicale, basta semplicemente moltiplicare la frequenza della nota, distante un semitono inferiore, per 1,0594631. Se si esegue la moltiplicazione 12 volte, si vedrà che la frequenza originale viene raddoppiata. In un’ottava musicale ci sono 12 semitoni (l’ottava è la distanza tra due note con lo stesso nome) pertanto, se raddoppiamo la frequenza, il suono sale di un’ottava. La tabella mostra la conversione da simboli musicali (la scala di ‘do centrale’) in frequenze audio.

I generatori d’inviluppo

L’inviluppo di un suono è la configurazione delle sue variazioni di volume sonoro: dall’istante in cui l’emissione inizia fino a quello in cui termina. Inviluppi simili a quelli di una nota di pianoforte e di altri strumenti sono mostrati nel diagramma.

Pianoforte naturale

Pianoforte artificiale

Gli inviluppi vengono comunemente suddivisi in 4 parti chiamate, di solito: Attacco, Decadimento, Prolungamento, Rilascio (ADSR). Nel caso del pianoforte, il volume, dopo che il tasto è stato battuto, sale rapidamente al suo livello più alto (attacco), quindi si abbassa lentamente (decadimento) ad un volume quasi costante mentre il tasto viene mantenuto abbassato (prolungamento) ed infine cade rapidamente a zero (rilascio) al momento che il tasto viene risollevato. Si noti che il 'prolungamento' è un livello di volume, mentre gli altri tre sono intervalli di tempo. Uno strumento che permette di controllare tutti i quattro gli aspetti di un inviluppo, viene chiamato "generatore ADSR". Il Commodore 64 rientra in questa categoria perché è capace di imitare gli inviluppi di molti comuni strumenti o di crearne dei nuovi "artificiali".

Le forme d'onda

Una forma d'onda è, appunto, la "forma" ripetitiva del segnale emessa da un oscillatore e determina il carattere di un particolare suono. Due strumenti diversi, che suonino note dello stesso tono, non emettono lo stesso suono perché, in parte, le forme d'onda prodotte non sono uguali. Le più comuni forme d'onda (nel 'regno' della musica) sono: l'onda quadra, l'onda triangolare e l'onda "a dente di sega", come mostrato nella figura.

Onda quadra **Onda triangolare** **Onda a dente di sega**

La maggior parte degli home computer produce una sola forma d'onda, di solito quella quadra, molto ricca di armoniche: questo spiega perché il suono generato è stridente. Attualmente il Commodore 64 è il computer più interessante perché è possibile selezionare a piacimento una delle tre forme d'onda di base su ciascuno dei tre oscillatori di cui dispone. Le forme d'onda possono venire modificate con filtri che alterano il tono (come succede negli apparecchi ad alta fedeltà agendo sui controlli bassi/alti). Questo permette di simulare suoni naturali con maggior fedeltà e di produrre nuovi suoni più stimolanti.

Il rumore

Il rumore è un tipo di suono complesso prodotto da vibrazioni casuali. Immaginiamo qualche suono comune come la pioggia, il vento ed il tuono. Questi rumori non sono simili tra loro in quanto prodotti da una combinazione di rumore puro (e quindi casuale) con qualche tono dominante. La maggior parte dei computer, con capacità di produrre rumore, permette quindi di modulare il rumore stesso o di mixarlo con note pure. La gamma di effetti possibili varia da "vento fischiante" a violente esplosioni.

Come programmare il Sound Interface Chip (SID) del Commodore 64

Tra gli home computer più diffusi, il Commodore 64 possiede le caratteristiche più sofisticate per la creazione di suoni, frutto dell'impiego di un chip specializzato chiamato Sound Interface Device (Unità d'Interfaccia per il Suono) da cui la sigla SID. Il SID possiede caratteristiche simili a quelle dei sintetizzatori monofonici comunemente in commercio. Ci sono tre oscillatori con una serie di otto ottave (da 0 a 3.900 Hz, in 65536 passi complessivi), un controllo principale di volume (da 0 a 15), quattro forme d'onda per ciascun oscillatore (triangolare, a dente di sega, ad impulso variabile e rumore), sincronizzazione dell'oscillatore e generatori di inviluppo che permettono di effettuare un controllo ADSR su ciascun oscillatore. Altre caratteristiche comprendono una modulazione "ad anello", un filtro programmabile (passa-basso, passa-banda e passa-alto) con il quale è possibile selezionare ed eliminare una ristretta banda di frequenze, un filtraggio d'inviluppo e

due interfacce analogico/digitali a potenziometro per controllare le funzioni del SID. Un ingresso audio esterno offre la possibilità di collegare alla macchina altri chip SID per cui, i segnali audio esterni possono essere immessi, filtrati e miscelati insieme agli output standard del SID.
È piuttosto difficile descrivere con poche parole gli effetti ottenibili con tutte queste funzioni: tenteremo di chiarire, a grandi linee, il significato di quanto appena esposto. Innanzitutto la sincronizzazione dell'oscillatore produce due segnali (in questo caso due particolari "timbri" o voci) che vengono armonizzati tra loro per creare un suono singolo e più complesso.
La modulazione, invece, è l'alterazione di un segnale ad opera di un secondo segnale che influisce sulla frequenza o sull'ampiezza (volume) del suono. La modulazione ad anello è la modulazione di ampiezza di un timbro, o voce, in dipendenza da una seconda voce: ciò produce un suono limpido ma con effetto stridente, dissonante ed utilizzabile, ad esempio, per simulare il suono di un campanello o di un tamburo di latta.
I filtri permettono di eliminare (tosare) determinati gruppi di frequenze da un segnale. I diversi tipi di filtraggio possibili sul Commodore 64 producono effetti che spiegano i loro stessi nomi: i filtri passa-basso escludono le frequenze superiori ad una data frequenza, i filtri passa-banda escludono le frequenze che si trovano al di sopra ed al di sotto di una determinata gamma di frequenze, i filtri d'assorbimento producono il risultato opposto ai filtri passa-banda, i filtri passa-alto eliminano le frequenze inferiori ad una data frequenza, la risonanza variabile può essere applicata a tutti i filtri finora esaminati per accentuare le frequenze che precedono e seguono i punti d'esclusione. Il filtraggio d'inviluppo rappresenta un caso particolare: questo produce un effetto diverso dai precedenti in quanto, i valori ADSR codificati numericamente e selezionati per l'inviluppo 3, possono essere applicati ad un segnale in modo da trasformare la struttura armonica durante la stessa emissione del suono. L'effetto è simile a quello prodotto da un filtro variabile.
Per generare anche semplici effetti sonori, è quindi necessario scrivere una serie di istruzioni BASIC e, in alcuni casi, questo linguaggio non garantisce una velocità tale da far completa giustizia dell'intera gamma di possibilità sonore del SID. Una descrizione completa dei registri di controllo, presenti nel chip, richiederebbe molto più tempo; chi volesse intraprendere un'attività "musicale" su questo computer, il mio consiglio è di guardarsi attorno alla ricerca di manuali specializzati in questo campo.
Nonostante ciò, a titolo d'esempio, riporto un breve programma dimostrativo (vedi sotto). Sebbene detto programma sia composto da ben 22 linee, esso permette di suonare, su di un oscillatore, solo cinque note di un semplice accordo. La linea 20 interrompe il collegamento tra il filtro e gli oscillatori; la linea 30 seleziona il volume principale al suo massimo valore; le linee 40 e 50 definiscono un inviluppo del tipo a "pianoforte"; la linea 80 seleziona la frequenza della nota; le linee 90 e 100 avviano ed interrompono il ciclo d'inviluppo ADSR e selezionano la forma d'onda a dente di sega per la voce 1.

```
 10 SID=54272
 20 POKE SID+23,0
 30 POKE SID+24,15
 40 POKE SID+5,40
 50 POKE SID+6,201
 60 FOR N=1 TO 5
 70 READ FH,FL,D
 80 POKE SID+1,FH: POKE SID,FL
          : REM * SUONA LA NOTA *
 90 POKE SID+4,33
100 FOR I=1 TO 300*D: NEXT I
110 POKE SID+4,32
120 FOR I=1 TO 100: NEXT I
130 NEXT N
140 FOR I=1 TO 200: NEXT I
150 POKE SID+24,0
160 REM **DATI**
170 DATA 57,172,1
180 DATA 64,188,1
190 DATA51,97,1
200 DATA 25,177,1
210 DATA 38,126,1
220 END
```

Di solito i suoni prodotti dal Commodore 64 vengono inviati direttamente al televisore attraverso l'interfaccia ad alta frequenza, ma si può anche collegare l'uscita audio ad un sistema hi-fi per ottenere riproduzioni di alta qualità. Esaminiamo, quindi, i principi coinvolti nella progettazione di un software che permetta di trasformare il Commodore 4 in una 'batteria elettronica'. Ovviamente la qualità dei risultati è migliore se viene impiegato un sistema di riproduzione ad alta fedeltà.
Oltre a produrre sofisticati suoni sotto il controllo del software, il chip SID può ricevere segnali audio generati da una periferica elettronica (eventualmente dotata di altri chip SID) o da strumenti musicali come chitarre elettriche. Il segnale ricevuto può essere "mixato" con l'uscita audio del chip SID ed elaborato dai suoi filtri. Le interfacce di I/O, però, vanno utilizzate con estrema attenzione poiché un errato collegamento delle linee esterne può danneggiare gravemente il computer. Prima di relizzare qualsiasi collegamento esterno, sarà meglio consultare sul manuale tecnico i dati relativi ai piedini ed ai livelli di tensione.
Per introdurre l'argomento, vediamo come sia possibile generare un suono musicale periodico, utilizzando una serie di note pure; in seguito verrà esaminato un metodo alternativo per ottenere lo stesso risultato.
Controllando l'inviluppo di una particolare nota periodica, è possibile, in diverse proporzioni, intodurre nel suono delle "armoniche". Ciò significa che si può creare qualsiasi suono periodico intervenendo su di un

ristretto numero di controlli, relativi all'inviluppo di un oscillatore, anche noti sotto la sigla ADSR. Sfortunatamente il BASIC del Commodore 64 non consente un sofisticato controllo dei suoni che, quindi, vanno programmati con un massiccio impiego di istruzioni PEEK e POKE. Se alla variabile SID si assegna il valore 54272 (l'indirizzo di base del chip SID), allora gli indirizzi compresi tra SID e SID+28 controlleranno il chip del suono e, di conseguenza, tutti i suoni generati sul Commodore 64.
Il programma in codice macchina che segue questo capitolo, trasforma il computer in una batteria elettronica a tre voci controllata da interrupt. Questa routine di "cuneo" può venire controllata dal BASIC ma, indipendentemente da questo, continuerà a suonare il ritmo e, quindi, con opportune modifiche, sarà possibile abbinarlo a qualsiasi programma BASIC per ottenere effetti sonori di sottofondo.

Suoni musicali

I suoni raggiungono l'orecchio umano come vibrazioni periodiche della pressione atmosferica. Il numero di vibrazioni per secondo è detto "altezza" o "frequenza" del suono. Il limite inferiore dell'udito umano è di circa 15 cicli al secondo (15 Hz); una nota pura con frequenza di 100 Hz, quindi, è appena percettibile, il LA superiore al DO centrale ha, per convenzione, una frequenza di 440 Hz. Se si raddoppia la frequenza di una nota, la sua altezza si innalza di un'ottava. Mediamente l'orecchio umano percepisce una gamma di suoni dell'estensione di circa 10 ottave. Gli oscillatori a tre note del SID, invece, possiedono una gamma di circa 8 ottave, approssimativamente da 0 a 4000 Hz.
Il fisico francese Jean Fourier (1768-1830) per primo osservò che qualsiasi forma d'onda periodica può essere considerata come composta da una nota pura fondamentale, amalgamata con note le cui frequenze sono multipli della nota fondamentale: tali note prendono il nome di "armoniche". Il timbro inconfondibile di un suono o di una nota dipende, dunque, dalle proporzioni tra le diverse frequenze armoniche che la compongono. Un'onda sinusoidale pura, che corrisponde ad una nota pura, è essenzialmente un segnale analogico difficilmente risproducibile con uno strumento digitale in cui sono presenti livelli di tensione di 0 e 5 volt. Per questo motivo, la tecnica adottata sui micro per ottenere un particolare suono periodico, non prevede la generazione di frequenze pure e la loro successiva fusione.
Ciascuna delle tre voci generabili dal chip SID può produrre 4 forme d'onda periodiche:

1. *Dente di sega:* in essa sono presenti tutte le armoniche. L'armonica "ennesima" possiede un'intensità proporzionale a 1/N.
2. *Triangolare:* contiene solo le armoniche dispari. Per N dispari, l'ennesima armonica possiede un'intensità proporzionale a $1/N^2$.
3. *Quadra:* quest'onda presenta armoniche dispari proporzionali a 1/N. Variando l'ampiezza dell'impulso si possono generare svariate onde rettangolari, ciascuna composta da un proprio amalgama di armoniche.
4. *Rumore bianco:* mistura casuale di frequenze usata per effetti particolari.

Il contenuto armonico di un suono può essere modificato con l'impiego dei filtri. Il chip SID offre tre tipi di filtro: passa-basso, passa-banda e passa-alto. Il primo, ad esempio, lascia passare tutte le frequenze minori di un valore prestabilito ed attenua tutte le frequenze superiori ad esso. Grazie a queste funzioni, ed ai controlli d'inviluppo ADSR, è possibile ottenere qualsiasi suono (anche una simulazione del parlato).
I programmi per la generazione del parlato si possono ottenere in più modi e qui ne viene descritto uno che abbina una discreta qualità ad un vocabolario illimitato. I suoi elementi di base, o "fonemi", vengono codificati in una matrice di valori ADSR. Un programma in codice macchina permette di tradurre i testi ASC (il codice alternativo del Commodore all'ASCII) in una sequenza di codici di fonema, successivamente inviati al chip SID per mezzo della matrice ADSR. Tale operazione è meno semplice di quanto possa sembrare poiché le norme, che regolano la trasposizione di un testo in fonemi, sono piuttosto complesse. La qualità del parlato finale dipende principalmente dalla qualità di questa sezione del programma. Comunque sia, questa applicazione è realizzabile sul Commodore 64 ed il mercato offre già numerosi pacchetti commerciali basati su di essa.

Controllo dell'inviluppo

Il diagramma ADSR a pagina seguente, rappresenta la forma d'onda generica di una nota musicale e ne evidenzia le caratteristiche controllabili con il chip SID. I fattori ADSR sono:

1. *Attacco:* il tempo di salita di una nota.
2. *Caduta:* il tempo che una nota impiega a ricadere ad un livello stabile.
3. *Persistenza:* il volume durante un livello stabile.
4. *Rilascio:* il tempo necessario affinché il volume di una nota passi dal livello di persistenza a 0.

Le sezioni di Attacco, Caduta e Rilascio di una nota vengono controllate da "nybble" (4 bit) interni ai registri del chip SID. Quindi ciascuno di questi parametri può assumere solo valori compresi tra 0 e 15 ed il rapporto, tra i loro valori, deve essere inserito (con POKE) nei registri del SID.

Inviluppi indirizzati
La qualità di una nota, ossia l'insieme delle caratteristiche che permettono di distinguere ad esempio il suono di un pianoforte da quello di un violino, dipende dalla forma o "profilo" dell'inviluppo. Nella sintetizzazione elettronica dei suoni, l'inviluppo si compone di quattro fasi diverse: Attacco, Caduta, Persistenza e Rilascio (ADSR, da Attack, Decay, Sustain e Release). La durata di ciascuna fase viene programmata inserendo (con POKE) particolari valori nei registri del chip SID. Questa tecnica consente di sintetizzare i suoni di diversi strumenti.

La seguente tabella contiene i valori effettivi di sincronizzazione:

Val.	Cadenza di Attacco	Cadenza di Caduta Persistenza
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 sec
11	800 ms	2.4 sec
12	1 sec	3 sec
13	3 sec	9 sec
14	5 sec	15 sec
15	8 sec	24 sec

La durata della fase di Persistenza dipende da un ciclo di ritardo. In base alla tabella di lato, i valori ADSR, per una nota di violino, dovrebbero essere:

ADSR	Tempo	Valori POKE
A	500 ms	10
D	300 ms	8
S	—	—
R	750 MS	9

Per produrre un suono sul Commodore 64, occorre programmare 6 operazioni diverse :

1) Accendere il volume con l'istruzione:
POKE SID+24,15
2) Selezionare l'ADSR. Ad esempio:
POKE SID+5,9: REM ATTACCO/CADUTA VOCE #1
POKE SID+6,0: REM PERSISTENZA/RILASCIO VOCE #1
3) Selezionare la frequenza di ogni oscillatore. Ad esempio:
POKE SID+1,25: REM BYTE-ALTO FREQUENZA VOCE #1
POKE SID,0: REM BYTE-BASSO FREQUENZA VOCE #1
4) Selezionare la forma d'onda prescelta. Ad esempio:
POKE SID+4,33: REM SELEZIONA L'ONDA A DENTE DI SEGA PER LA VOCE #1

A questo punto inizia l'emissione del suono, detta anche "apertura della porta".

5) Esecuzione di un ciclo di ritardo mentre la nota si trova a livello di Persistenza.
6) Rilascio della forma d'onda. Ad esempio:
POKE SID+4,32: REM RILASCIO DELL'ONDA A DENTE DI SEGA

Il modo più semplice per programmare l'emissione di una nota sul Commodore 64, è selezionare i valori iniziali di ADSR e costruire un ciclo FOR che legga (READ) da frasi DATA i byte alto/basso della frequenza. La

presenza di valori 0, in queste istruzioni, permette di modificare il ritmo delle diverse voci mantenendo costante la durata del ritardo.

BATTERIA ELETTRONICA

Il seguente programma usa un “cuneo” IRQ, in codice macchina, che continua a generare il suono di una batteria indipendentemente dal lavoro svolto dal programma BASIC. La sezione in codice macchina esamina, inoltre, il contenuto della locazione 197 ($00C5) per individuare quali tasti sono stati battuti. Detta locazione di pagina 0, contiene il valore relativo all’ultimo tasto premuto. Al posto di una chiamata al Kernel, può essere utilizzato un programma in codice macchina per la scansione della tastiera, a patto che $00C5 venga esaminato con sufficiente frequenza.
Poiché il programma pilota in BASIC non usa le istruzioni GET o INPUT per svuotare il buffer di tastiera, la routine in codice macchina deve, come ultima operazione, azzerare il puntatore nella locazione 198 ($00C6) in cui, sempre in pagina 0, abitualmente contiene il numero dei tasti battuti ed accumulati nel buffer della tastiera.
Durante l’esecuzione del programma, ad ogni tamburo vengono assegnati 16 “intervalli” di tempo, visualizzati sullo schermo, durante i quali esso può suonare o restare muto. La scelta del valore 16 facilita la composizione di ritmi rock in quattro quarti. Lo sprite del cursore viene quindi spostato sulla griglia degli intervalli e posizionato sulla cella desiderata; a questo punto basta premere il tasto Invio per attivare o disattivare un intervallo di tempo. Per avviare il ritmo selezionato, si preme il tasto F1. Il menu sullo schermo visualizza le scelte possibili. Poiché l’emissione dei suoni è controllata da interrupt, la combinazione ritmica può essere modificata sul video mentre il ritmo viene suonato.
Il programma BASIC contiene tutto il codice macchina necessario (sotto forma di frasi DATA) per far funzionare il programma e può essere trascritto ed eseguito senza alcuna modifica. Se si desidera utilizzare il listato in Assembly della parte del programma in codice macchina (vedere alla pagina seguente), occorre assemblarlo e poi eliminare dal programma BASIC le linee da 1520 a 1540 e le istruzioni DATA tra linee 1620 e 1940.
Dopo aver lanciato il programma, si provi a selezionare la seguente combinazione che genera un ritmo facilmente riconoscibile:

```
   ****  CBM64 RHYTHM GENERATORE   ****

TEMPO = 15

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
X _ _ _ X _ _ X X _ _ _ X _ _ _ -BASS
_ _ X _ _ _ X _ _ _ X _ _ _ X X -SNARE
_ X _ X _ X _ X _ X _ X _ X _ X -BELL
```

Programma BASIC generatore di ritmo

```
1000 REM **GENERATORE DI RITMO**
1010 PRINT CHR$(147): REM RIPULISCE LO SCHERMO
1020 GOSUB 1510: REM PREPARA CO. MACCHINA & SPRITE
1030 GOSUB 1270: REM PREPARA LO SCHERMO
1040 SYS(49152): REM * INSERISCE IL CUNEO *
1050 REM SYS(49175) PER RIMUOVERE IL CUNEO
1060 REM * CICLO PRINCIPALE *
1070 P=PEEK(197): REM ULTIMO TASTO BATTUTO
1080 IF p=37 THEN Y=Y-16: SY=SY-1: IF Y<95 THEN Y=95: SY=0
1090 IF P=36 THEN Y=Y+16: SY=SY+1: IL Y>127 THEN Y=127: SY=2
1100 IF P=47 THEN X=X-16: SX=SX-1: IF X<28 THEN X=28: SX=0
1110 IF P=44 THEN X=X+16: SX=SX+1: IF X>268 THEN X=268: SX=15
1120 Z=0: IF P=1 THEN Z=1
1130 PRINT CHR$(19); S$; “TEMPO=”; PEEK(679); CHR$(157); “   “
1140 POKE VIC+16,X/256: POKE VIC, X AND 255
1150 POKE VIC+1,Y
1160 REM * CALCOLA LA POSIZIONE DI SCHERMO OM BASE ALLE
          COORDINATE x, y DELLO SPRITE *
1170 SC=B+INT((Y-50)/8+1)*40+INT((X-18)/8)
1180 IF Y=0 THEN ROW=50000: REM TAMBURO BASSO
1190 IF S=1 THE ROW=50016: REM TAMBURO RULLANTE
1200 IF SY=2 THEN ROW=50032: REM CAMPANA
1210 IF Z=1 AND PEEK(SC)=32 THEN POKE SC,24: POKE ROW+SX,1:
          GOTO 1230
1220 IF Z=1 AND PEEK(SC)=24 THEN POKE SC,32: POKE ROW+SX,0
1230 IF P=51 THE GOSUB 1270: FOR I=0 TO 50: POKE 50000+I,0: NEXT
1240 IF P=57 THEN POKE 198,0: POKE VIC+21,0: PRINT CHR$(147): END
1250 GOTO 1070
1260 REM * PREPARA LO SCHERMO *
1270 PRINT CHR$(19)
1280 S$=CHR$(17)+CHR$(17)+CHR$(17)
1290 PRINT TAB(3)”                                      “: REM 38 SPAZI
1300 PRINT CHR$18)TAB(3)” **** GENERATORE DI RITMO CBM 64 **** “
1310 PRINT: PRINT: PRINT
1320 PRINT” - - - - - - - - - - - - - - - - “
1330 PRINT”                                 -BASSO”
1340 PRINT” - - - - - - - - - - - - - - - - “
1350 PRINT”                                 -RULLANTE”
1360 PRINT” - - - - - - - - - - - - - - - - “
1370 PRINT”                                 -CAMPANA”
1380 PRINT” - - - - - - - - - - - - - - - - “
1390 PRINT : PRINT TAB(8) ”OPZIONI...... ”
1400 PRINT TAB(8) ”[F1]       SUONA UN RITMO”
1410 PRINT TAB(8) ”[F3]       RIDUCE IL TEMPO”
1420 PRINT TAB(8) ”[F5]       AUMENTA IL TEMPO”
1430 PRINT TAB(8) ”[F7]       FERMA IL RITMO”
1440 PRINT TAB(8) ”[CLR]      CANCELLA LA GRIGLIA »
1450 PRINT TAB(8) ”[T]        FINE DEL PROGRAMMA”
1460 PRINT TAB(8) ”[<]        CURSORE A SINISTRA”
1470 PRINT TAB(8) ”[>]        CURSORE A DESTRA”
1480 PRINT TAB(8) ”[K]        CURSORE IN ALTO”
1490 PRINT TAB(8) ”[M]        CURSORE IN BASSO”
1500 RETURN
1510 REM * PREPARA COD. MACCHINA & SPRITE *
1520 FOR I=49152 TO 49413
1530 READ J: C=C+J: POKE I,J: NEXT I
1540 READ J: IF C<>J THEN PRINT ”DATA ERROR”: END
1550 FOR I=0 TO 62: READ J: POKE 832+I,J: NEXT I
1560 VIC=53248: X=28: Y=95: B=1024: SID=54272
1570 FOR I=0 TO 24 : POKE SID+I,0: NEXT I
1580 POKE SID+24,15: POKE VIC+21,1: POKE 2040,13
1590 POKE 254,15: POKE 679, 15: POKE VIC+39,1
1600 FOR I=0 TO 50: POKE 50000+I,0: NEXT I
1610 RETURN
1620 REM ***ISTRUZIONI DATA DEL COD. MACCHINA***
1630 DATA 120,173,20,3,133,251,173,21,3
1640 DATA 133,252,169,36,141,20,3,169
1650 DATA 192,141,21,3,88,96,120,165,251
1660 DATA 141,20,3,165,252,141,21,3,88
1670 DATA 96,32,177,192,165,197,201,3
1680 DATA 208,3,32,135,192,165,197,201,4
1690 DATA 208,3,32,140,192,165,197,201,5
1700 DATA 208,3,32,155,192,165,197,201,6
1710 DATA 208,3,32,166,192,32,87,192,142
1720 DATA 168,2,140,169,2,76,49,234,165
1730 DATA 253,208,1,96,198,254,240,1,96
1740 DATA 32,149,192,224,16,208,3,32,144
1750 DATA 192,185,80,195,240,3,32,184
1760 DATA 192,185,96,195,240,3,32,210
1770 DATA 192,185,112,195,240,3,32,236
1780 DATA 192,200,234,232,96,169,0,133
1790 DATA 253,96,169,1,133,253,162,0,160
1800 DATA 0,96,173,167,2,133,254,96,173
1810 DATA 167,2,201,255,240,3,238,167,2
1820 DATA 96,173,167,2,201,1,240,3,206
1830 DATA167,2,96,174,168,2,172,169,2
1840 DATA 96,169,14,141,6,212,169,32,141
1850 DATA 2,212,169,66,141,4,212,169,3
1860 DATA 141,1,212,169,65,141,4,212,96
1870 DATA 169,7,141,12,212,169,12,141,13
1880 DATA 212,169,128,141,11,212,169,65
1890 DATA 141,8,212,169,129,141,11,212
1900 DATA 96,169,2,141,19,212,169,13,141
1910 DATA 20,212,169,18,141,18,212,169
1920 DATA 100,141,15,212,169,17,141,18
1930 DATA 212,96
1940 DATA 32038: REM * CHECKSUM *
1950 REM ***ISTRUZIONI DATA DELO SPRITE (CURSORE)***
1960 DATA 127,254,0,127,254,0,127,254,0
1970 DATA 112,14,0,112,14,0,112,14,0,112
1980 DATA 14,0,112,14,0,112,14,0,112,14
1990 DATA 0,112,14,0,127,254,0,127,254,0
2000 DATA127,254,0,0,0,0,0,0,0,0,0,0,0
2010 DATA 0,0,0,0,0,0,0,0,0,0,0
```

Listato Assembly

```
;++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++
;++                            ++
;++       CODICE SORGENTE      ++
;++  PER BATTERIA ELETTRONICA  ++
;++                            ++
;++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++
;
VOL            = $D4181; SID volume
ATT1           = $D405  ; SID attacco voce 1
PERS1          = $D406  ; SID persistenza voce 1
IMPULSO        = $D402  ; SID freq. d'impulso voce 1
ONDA1          = $D404  ; SID forma d'onda voce 1
BASSO          = $D401  ; SID byte-alto della freq. voce 1
ATT2           = $D40C  ; SID attacco voce 2
PERS2          = $D40D  ; SID persistenza voce 2
ONDA2          = $D40B  ; SID forma d'onda voce 2
RULLANTE       = $D408  ; SID byte-alto della freq. voce 2
ATT3           = $D413  ; SID attacco voce 3
PERS3          = $D414  ; SID persistenza voce 3
ONDA3          = $D412  ; SID forma d'onda voce 3
CAMPANA        = $D40F  ; SID byte-alto della freq. voce 3
RIGA1          = $C350  ; deposito per voce 1
RIGA2          = $C360  ; deposito per voce 2
RIGA3          = $C370  ; deposito per voce 3
TEMPO          = $02A7  ; deposito temporaneo per il ritardo
XCONT          = $02AB  ; deposito temporaneo per reg. X
YCONT          = $02A9  ; deposito temporaneo per reg. Y
BBVETT         = $FB    ; deposito per byte-basso vettore
BAVETT         = $FC    ; deposito per byte-alto vettore
SUONA          = $FD    ; innesco per emissione del suono (1=si)
RITARDO        = $FE    ; deposito per ritardo corrente
TASTO          = $C5    ; ultimo tasto premuto
;---------------------------------------------------------------------
*              = $C000  ; assembla da 49152 (decimale)
;---------------------------------------------------------------------
;PREPARA IL CUNEO
;---------------------------------------------------------------------
        SEI             ; disabilita la richiesta d'interruzione
        LDA $0314       ; riceve contenuto del byte-basso del vett.
        STA BBVETT      ; memorizza in BBVETT
        LDA $0315       ; riceve contenuto del byte-alto del vett.
        STA BAVETT      ; memorizza in BAVETT
        LDA #<CUNEO     ; riceve il byte-basso dell'inizio del cuneo
        STA $0314       ; memorizza bel byte-basso del vett. IRQ
        LDA #>CUNEO     ; riceve byte-alto dell'inizio del cuneo
        STA $0315       ; memorizza nel byte-alto del vett. IRQ
        CLI             ; riabilita la richiesta d'interruzione
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
;RIMUOVE IL CUNEO
;---------------------------------------------------------------------
        SEI             ; disabilita la richiesta d'interruzione
        LDA BBVETT      ; riceve il valore iniziale di BBVETT
        STA $0314       ; registra nel byte-basso del vett. IRQ
        LDA BAVETT      ; riceve il valore iniziale di BAVETT
        STA $0315       ; registra nel byte-alto del vett. IRQ
        CLI             ; riabilita la richiesta d'interruzione
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
;CICLO PRINCIPALE
;---------------------------------------------------------------------
CUNEO   JSR REG         ; passa il controllo a REG
        LDA TASTO       ; quale tasto è stato premuto?
        CMP #$03        ; è il tasto di funzione #1?
        BNE CONT1       ; salta se "non vero"
        JSR FLAG0       ; passa il controllo a FLAG0
CONT1   LDA TASTO       ; quale tasto è stato premuto?
        CMP #$04        ; è il tasto di funzione #7?
        BNE CONT2       ; salta se "non vero"
        JSR FLAG1       ; passa il controllo a FLAG1
CONT2   LDA TASTO       ; quale tasto è stato premuto?
        CMP #$05        ; è il tasto di funzione #3?
        BNE CONT3       ; salta se "non vero"
        JSR SOMMA       ; passa il controllo a SOMMA
CONT3   LDA TASTO       ; quale tasto è stato premuto?
        CMP #$06        ; è il tasto di funzione #5?
        BNE CONT4       ; salta se "non vero"
        JSR SOTTR       ; passa il controllo a SOTTR
CONT4   JSR REST        ; passa il controllo a REST
        STX XCONT       ; memorizza il valore del reg. X
        STY YCONT       ; memorizza il valore nel reg. Y
        JMP $EA31       ; torna al punto d'interruzione
;---------------------------------------------------------------------
;ROUTINE DI VERIFICA
;---------------------------------------------------------------------
REST    LDA SUONA       ; riceve il valore d'innesco
        BNE INIZIO      ; se uguale a 1, salta
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
INIZIA  DEC RITARDO     ; riduce il ritardo
        BEQ AVVIA       ; salta se 0
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
AVVIA   JSR CONT        ; passa il controllo a CONT
        CPX #$10        ; fine del ciclo?
        BNE VERIF       ; salta se non 0
        JSR AZZERA      ; passa il controllo a AZZERA
;---------------------------------------------------------------------
VERIF   LDA RIGA1,Y     ; riceve l'offset di RIGA1 da Y
        BEQ SUCC1       ; salta se 0
        JSR BATT1       ; passa il controllo a BATT1
SUCC1   LDA RIGA2,Y     ; riceve l'offset di RIGA2 da Y
        BEQ SUCC2       ; salta se 0
        JSR BATT2       ; passa il controllo a BATT2
SUCC2   LDA RIGA3,Y     ; riceve l'offset di RIGA3 da Y
        BEQ SUCC3       ; salta se 0
        JSR BATT3       ; passa il controllo a BATT3
SUCC3   INY             ; incrementa l'offset
        INX             ; incrementa il contatore di ciclo
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
;SUBROUTINE
;---------------------------------------------------------------------
FLAG0   LDA #$00        ; registra 0
        STA SUONO       ; in suono
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
FLAG1   LDA #$01        ; registra 1
        STA SUONO       ; in suono
AZZERA  LDX #$00        ; azzera il reg. X
        LDY #$00        ; azzera il reg. Y
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
CONT    LDA TEMPO       ; riceve il valore di TEMPO
        STA RITARDO     ; lo registra in RITARDO
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
SOMMA   LDA TEMPO       ; riceve il valore di TEMPO
        CMP #$FF        ; confronta il risultato con 255
        BEQ CONT5       ; salta se "vero"
        INC TEMPO       ; incrementa tempo
CONT5   RTS             ; restituisce il controllo
;---------------------------------------------------------------------
SOTTR   LDA TEMPO       ; riceve il valore di TEMPO
        CMP #$01        ; confronta il risultato con 1
        BEQ CONT6       ; salta se "vero"
        DEC TEMPO       ; decrementa tempo
CONT6   RTS             ; restituisce il controllo
;---------------------------------------------------------------------
REG     LDX XCONT       ; memorizza il valore di XCONT nel reg. X
        LDY YCONT       ; memorizza il valore di YCONT nel reg. Y
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
;ROUTINE DI SUONO DELLA BATTERIA
;---------------------------------------------------------------------
BATT1   LDA #$0E        ; ⎫
        STA PERS1       ; ⎪
        LDA #$20        ; ⎪
        STA IMPULSO     ; ⎪
        LDA #$42        ; ⎬  prepara il tamburo
        STA ONDA1       ; ⎪  basso e lo suona
        LDA #$03        ; ⎪
        STA BASSO       ; ⎪
        LDA #$41        ; ⎪
        STA ONDA1       ; ⎭
        RTS             ; restituisce il controllo
;---------------------------------------------------------------------
BATT2   LDA #$07        ; ⎫
        STA ATT2        ; ⎪
        LDA #$0C        ; ⎪
        STA PERS2       ; ⎬  prepara il tamburo
        LDA #$80        ; ⎪  rullante e lo suona
        STA ONDA2       ; ⎪
        LDA #$41        ; ⎪
        STA ONDA2       ; ⎪
        RTS             ; ⎭
```

```
;-------------------------------------------------------------------------------------
BATT3   LDA #$02        ;
        STA ATT3        ;
        LDA #$0D        ;
        STA PERS3       ;
        LDA #$12        ;     prepara la campana
        STA ONDA3       ;     e la suona
        LDA #$64        ;
        STA CAMPANA     ;
        LDA #$11        ;
        STA ONDA3       ;
        RTS             ; restituisce il controllo
        .END
```

Disegni reticolari

Esploriamo le routine in virgola mobile del BASIC

Sfortunatamente le routine in virgola mobile, dell'interprete BASIC per il Commodore 64, non sono affatto ben documentate nei vari manuali. Ad esempio, non viene riprodotta una tabella dei salti (come quella del kernel) che permetta di accedere facilmente ad esse. Il progetto, presentato in questa prima parte dedicata all'argomento, riguarda il tracciamento e la rotazione di disegni sullo schermo ad alta risoluzione del Commodore 64.
Gli accorgimenti e le teorie matematiche qui esposte, comunque, possono venir fruttuosamente impiegate anche per ottenere routine aritmetiche più veloci (ad esempio per la moltiplicazione di matrici numeriche). Si risordi, però, che non sempre conviene eseguire calcoli matematici in tempo reale, specialmente quando si tratta di tracciare disegni in rapida sequenza sullo schermo: spesso conviene eseguire i calcoli in precedenza, memorizzarne i risultati e, infine, passare al disegno animato. Ai nostri fini, tuttavia, il calcolo in tempo reale è il più adatto.
Le varabili del BASIC sono conservate, in memoria, sopra il programma BASIC. L'indirizzo iniziale della tabella delle variabili è contenuto in un puntatore che si trova nelle locazioni 45 e 46 (decimale). Come tutti i puntatori nel Commodore 64, anche questo è conservato nel formato basso/alto. Pertanto la formula per ricavare l'indirizzo della tabella è:

```
PEEK(45)+256*PEEK(46)
```

I puntatori associati alle variabili, insieme al loro normale contenuto, sono riassunti qui:

Punatatore	Funzione	Contenuto normale
43/44	Inizio del BASIC	2049
45/46	Inizio variabili	Dipende dalla lunghezza del programma
47/48	Inizio delle matrici	Dipende dal numero di variabili
49/50	Fine matrici + 1	Dipende dal numero e dalle dimensioni delle matrici
51/52	Fondo area stringhe	Dipende dal numero e dalla lunghezza delle stringhe
55/56	Limite superiore memoria	40960

Le stringhe, gestire dinamicamente, crescono verso il basso, partendo dal limite superiore della memoria, via via che vengono definite. Le matrici, invece, vengono conservate sopra la tabella delle variabili così, quando nel programma viene definita una nuova variabile, il sistema operativo sposta l'intera area delle matrici di tanti byte quanti ne occorrono per far entrare la nuova variabile.
La conservazione delle stringhe è necessariamente più complessa rispetto alle altre variabili. Infatti le variabili intere (se non appartengono ad una matrice) e le variabili a virgola mobile, occupano sette byte, ma una stringa può occuparne fino a 255. Per superare questa complicazione, il sistema operativo conserva, nella tabella delle variabili, soltanto la lunghezza della stringa ed un puntatore all'indirizzo dove viene conservata in memoria. Se la stringa è definita nel programma BASIC (ad esempio: A$="ABCD") allora il puntatore contiene, nell'area del programma BASIC, l'indirizzo del primo byte della stringa. Una simile stringa viene chiamata "statica". Se e quando il programma altera il contenuto di una stringa, allora questa viene chiamata "dinamica" e viene conservata nell'area subito sottostante il limite superiore della memoria. Ovviamente l'indirizzo del puntatore, nella tabella delle variabili, riflette la nuova posizione di tale stringa. Solo così il formato degli elementi nella tabella rimane costante (sette byte).

L’indirizzo di una stringa può essere estratto dalla tabella delle variabili usando un piccolo trucco e poche istruzioni. Infatti, subito dopo l’uso di una variabile BASIC, le locazioni 71 e 72 (decimali) in pagina zero, contengono l’indirizzo di tale variabile. Comunque sia, il recupero di questo indirizzo deve essere immediato poiché il sistema operativo utilizza la locazione 71 per certe operazioni aritmetiche. Il seguente programma mostra il formato usato per conservare in memoria le variabili ordinarie. La prima routine estrae una stringa dalla memoria e conserva immediatamente il valore del puntatore in 71 e 72 depositandolo in due locazioni del buffer per il registratore. In seguito l’indirizzo salvato viene usato per calcolare quello della stringa.

```
1000 REM **TROVA UNA STRINGA IN MEMORIA**
1010 X$=”ABCDEF”
1020 REM **X$** DIVENTA VARIABILE CORRENTE
1030 X$=X$+”“
1040 REM **SALVA IL PUNTATORE ALLA TABELLA**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM **INDIRIZZO NELLA TABELLA**
1070 INDR=PEEK(828)+256*PEEK(829)
1080 REM **ESAMINA ELEM. IN TABELLA**
1090 LS=PEEK(INDR):REM LUNGH. STRINGA
1100 SA=PEEK(INDR+1)+256*PEEK(INDR+2)
1110 REM SA = INDIRIZZO INIZ. DELLA STRINGA
1120 REM **ADESSO LEGGE LA STRINGA**
1130 FOR I=SA TO SA+LS
1140 VAR$=VAR$+CHR$(PEEK(I))
1150 NEXT
1160 PRINT VAR$
```

Qualcuno potrebbe pensare che, utilizzando variabili intere (quelle con il simbolo %), si risparmierebbe memoria ed i calcoli sarebbero più rapidi. Sul Commodore 64 ciò non è vero poiché, per eseguire calcoli sugli interi, questi vengono prima convertiti in virgola mobile e poi richiamate le routine per il calcolo in virgola mobile! Pertanto, sebbene le variabili intere occupino solo due byte (salvo quando fanno parte di una matrice), ad esse vengono assegnati comunque sette byte di memoria e, quelli in eccesso, vengono semplicemente ignorati nel corso del calcolo. La seguente routine consente di individuare in memoria una variabile intera:

```
1000 REM **TROVA UN INTERO IN MEMORIA**
1010 X%=3456
1020 REM **RENDE X% VARIABILE CORRENTE**
1030 X%=X%
1040 REM **SALVA IL PUNTATORE**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM **INDIRIZZO IN TABELLA VARIABILI**
1070 INDR=PEEK(828)+256*PEEK(829)
1080 REM **ESAMINA ELEM. IN TABELLA**
1090 LO=PEEK(INDR+1):HI=PEEK(INDR)
1100 REM **CALCOLA RISULTATO**
1110 BITSEGN=(HI AND 128)/128
1120 VAR=LO+256*(HI AND 127)-32768*BITSEGN
1130 PRINT VAR
```

La stessa tecnica di individuazione può essere impiegata per localizzare variabili in virgola mobile. Tuttavia, per far ciò, esiste un metodo più economico basato sul fatto che, usando DEF FN per definire una funzione della variabile X, tale variabile viene usata (pur rimanendo inalterata) ad ogni chiamata della funzione.
Questo programma utilizza DEF FN per calcolare l’indirizzo dell’attuale variabile BASIC, sempre ricorrendo alle locazioni 71 e 72. Poiché X viene usata come variabile per la funzione, si può esser certi che l’indirizzo risultante è quello del primo byte nella tabella delle variabili. Per assegnare l’indirizzo INDR viene quindi chiamata la funzione FN. Si noti che, passando uno zero (o un altro valore) attraverso il comando FN, nella tabella non cambia né il valore di X né il suo indirizzo.

```
1000 REM **TROVA VARIABILE A V.M.**
1010 DEF FNINDR(X)=PEEK(71)+256*PEEK(72)
1020 INDR=FNINDR(0):REM FORNISCE SEMPRE L’INDIRIZZO DI X
1030 X=-3.14159
1040 REM **CONVERTE DA BASE 2 A DECIMALE**
```

```
1050 POTENZA=2PEEK(INDR)-129)                    ?
1060 SEGNO=(-1)PEEK(INDR+1)AND128)/128)          ?
1070 REM **PARTE FRAZIONARIA LUNGA 31 BIT**
1080 D1=PEEK(INDR+1)AND127:REM 7 BIT
1090 D2=PEEK(INDR+2):REM 8 BIT
1100 D3=PEEK(INDR+3):D4=PEEK(INDR+4)
1110 REM **GULP!**
1120 FRAZ=2^(-7)*D1+2^(-15)*D2+2^(-23)*D3+2^(-31)*D4
1130 MANT=1+FRAZ
1140 VAR=SEGNO*POTENZA*MANT
1150 PRINT VAR
```

Matrici in memoria

Quando il BASIC esegue un'istruzione DIM, viene riservata memoria per la matrice specificata. Il blocco di memoria consiste in una "testata" più il numero di byte necessari per la conservazione degli elementi. Il formato di questi elementi varia a seconda del tipo di variabile fin qui esaminati (vedi diagramma). Se si intende accedere agli elementi della matrice usando il codice macchina, è molto importante comprendere quali siano le differenze.

Tipi di variabile

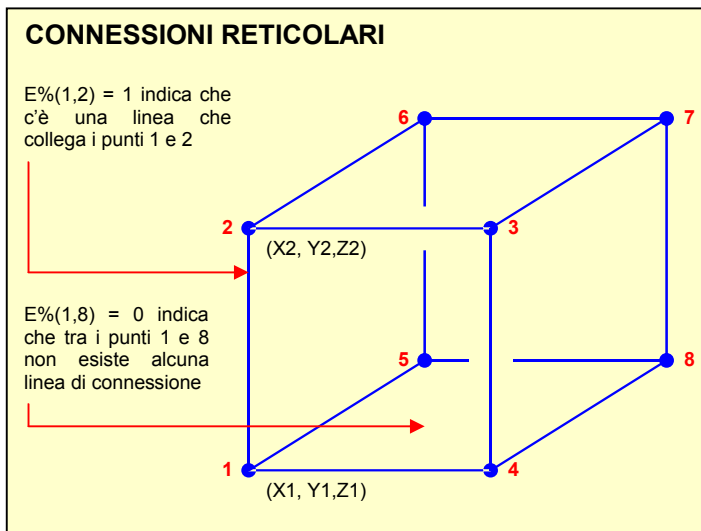
Sul Commodore 64 esistono tre tipi di variabili, ognuna delle quali ha un diverso formato quando conservato in memoria. Le variabili intere vengono conservate in due byte sotto forma di numeri in complemento a due. I numeri in virgola mobile (o fluttuante) richiedono sette byte: cinque per la mantissa e due per il segno e per l'esponente. Le varibili stringa vengono conservate, in una zona diversa della memoria, usando un byte per ciascun carattere della stringa. Però, questi dati, non vengono depositati nella tabella delle matrici, ma solo tre byte: uno per indicare la lunghezza della stringa e due che forniscono l'indirizzo a 16 bit nel quale viene memorizzata la stringa.

Inoltre occorre esaminare anche il meccanismo del calcolo in virgola mobile. Quando l'interpre BASIC esegue calcoli questo tipo di calcolo, eventuali risultati intermedi vengono depositati in due "accumulatori a virgola mobile" chiamati, di solito, FAC e ARG. Il formato è identico ai numeri in virgola mobile conservati in memoria. FAC si trova agli indirizzi da \$61 a \$65 (da 97 na 101 decimale) ed ARG da \$69 a \$6D (da 105 a 109 decimale). Per semplicità, in seguito, vengono utilizzate solo le routine dell'interprete che scmbiano numeri tra FAC e la memoria. Le routine dell'interprete BASIC, con le quali familiarizzarsi in questa prima parte, sono:

- MOVFM (indirizzo \$BBA2)
  Questa routine copia in FAC un numero in virgola mobile che si trova in memoria. Simbolicamente viene rappresentato con F←M. Per chiamare la routine occorre caricare, nell'accumulatore, il byte-basso dell'indirizzo per il numero in memoria e, nel registro Y, il byte alto.
- MOVMF (indirizzo \$BBD4)
  Questa routine deposita il contenuto di FAC in sette byte consecutivi della memoria. Simbolicamente viene rappresentato con M←F. Per la chiamata occorre caricare nei registri X ed Y, rispettivamente, il byte-basso e quello alto dell'indirizzo di memoria ove depositare il numero
- FMULT (indirizzo \$BA28)
  Questa è la routine di moltiplicazione: essa moltiplica il contenuto di FAC per un altro numero che si trova in memoria, depositando il risultato in FAC. Il primo numero viene caricato in FAC usando MOVFM mentre, il

byte-basso quello alto, vanno posti rispettivamente nei registri X ed Y prima di chiamare questa routine. Il risultato viene depositato in FAC ma può essere trasferito in memoria usando MOVMF.

- FADD (indirizzo $B867)
  Questa routine esegue la somma FAC=MEM+FAC. Per chiamarla, presupponendo che in FAC sia già presente un valore, occorre depositare, nell'accumulatore e nel registro Y[U], rispettivamente il byte-basso e quello alto di MEM.
- FSUB (indirizzo $B850)
  Questa routine esegue la differenza FAC=MEM-FAC. La chiamata si esegue come descritto per FADD.

Queste sono le routine dell'interprete utilizzate nella formulazione della prima parte dei programmi di grafica reticolare.
Il concetto base, del nostro progetto grafico, è che una figura reticolare può venire definita per mezzo di una serie di numeri (o nodi) ed una matrice di connessioni. I nodi hanno coordinate X(I), Y(I) e Z(I) dove I varia da 1 a NP (il numero di punti). La matrice delle connessioni è E%(I,J), dove I e J variano entrambi da 1 a NP.

L'elemento E%(I,J) vale 1 se il punto I è unito da un tratto al punto J, altrimenti vale zero. Questo metodo non è tra i più economici in fatto di memoria poiché si usano due byte quando, in realtà, ne basterebbe uno solo; tuttavia facilita la definizione dei punti collegati tra loro. Inoltre, nella pratica, NP non sarà mai molto grande.
Per far comunicare il BASIC con la routine in codice macchina che esegue la rotazione, si usano delle PEEK con le quali depositare gli indirizzi delle matrici, iniziando dall'elemento 1. Per cui, prima di tutto, occorre trovare gli indirizzi di X(1), Y(1), Z(1) ed E%(1,1). Al codice macchina, inoltre, viene passato il numero di punti, NP, ed il valore di COS e SIN dell'angolo di rotazione attorno all'asse Z desiderato.

È bene che qualsiasi progetto in codice macchina proceda per piccoli passi, altrimenti il compito di individuare eventuali errori diventa impossibile. Il progetto è stato, così, suddiviso in tre parti. Nella prima vengono definiti gli algoritmi e redatta una versione di prova in BASIC che ruota, attorno all'asse Z, un cubo tridimensionale proiettando i risultati sui piani X ed Y.
L'obiettivo, in seguito, sarà quello di convertire il programma BASIC in codice macchina. Anche questo avviene gradualmente: per prima viene convertita la subroutine che inizia alla linea 1800. Il risultato è il listato I-ROTSUB per il quale viene fornito anche il relativo programma di prova.
Nella seconda ed ultima parte dedicata a questo argomento, viene completato il programma in codice macchina per la rotazione di immagini reticolari e fornito il relativo programma caricatore in BASIC.

PROGRAMMA I-ROTSUB 64

Il seguente listato Assembly va assemblato e convertito in codice con l'apposito programma di load.
Si assegni al file risultante il nome "I-ROT.HEX".

```
; +++++++++++++++++++++++
; ++       I-ROTSUB 64      ++
; ++                        ++
; +++++++++++++++++++++++
; ++ USA MATRICI DEFINITE++
; ++ DAL BASIC. ESEGUIRE ++
; ++ POKE DEGLI INDIRIZZI ++
; +++++++++++++++++++++++
;
; PLTSUB          = $C183
; XLO             = $C103
; XHI             = $C104
; YLO             = $C105
;
*                 = $C544
;
; +++  VARIABILI ROTSUB  +++
;
; VARIABILI CHIAMATE DAL BASIC
XBASLO  *=*+1     ; POKE50500,X(0)LO
XBASHI  *=*+1     ; POKE50501,X(0)HI
YBASLO  *=*+1     ; POKE 50502,Y(0)LO
YBASHI  *=*+1     ; POKE 50503,Y(0)HI
NP      *=*+1     ; POKE50504,NP
; VARIABILI USATE DAL C/M
XILO    *=*+1
XIHI    *=*+1
YILO    *=*+1
YIHI    *=*+1
CSLO    *=*+1
CSHI    *=*+1
SNLO    *=*+1
SNHI    *=*+1
MEM1    *=*+5     ; var. virgola mobile
MEM2    *=*+5     ; var. virgola mobile
FAC      =$0061
ARG      =$0069
;
; ROUTINE ARITMETICHE DELL'INTERPRETE
;
FMULT    =$BA28  ; FAC=FAC*ARG
FADDT    =$b86A  ; FAC=FAC+ARG FSUB        =$B850  ;
FAC=MEM-FAC
FADD     =$B867  ; FAC=FAC+MEM
MOVFM    =$BBA2  ; FAC=MEM
MOVMF    =$BBD4  ; MEM=FAC
;
; +++ SALVA I REGISTRI +++
;
        PHA
        TXA
        PHA
        TYA
        PHA
;
; +++ INIZIALIZZA VARIABILI +++
;
        LDA XBASLO
        STA XILO
        LDA XBASHI
        STA XIHI
        LDA YBASLO
        STA YILO
        LDA YBASHI
        STA YIHI
;
; +++ ESEGUE MEM1=X(I)*CS-Y(I)*SN +++
;
INIZIO
        LDA XILO
        LDY XIHI
        JSR MOVFM          ; FAC=X(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT          ; FAC=X(I)*CS
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF          ; MEM1=X(I)*CS
        LDA YILO
        LDY YIHI
        JSR MOVFM          ; FAC=Y(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT          ; FAC=Y(I)*SN
        LDA #<MEM1
        LDY #>MEM1
        JSR FSUB           ; FAC=MEM1-FAC
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF          ; MEM1=FAC
;
; +++ ESEGUE MEM2=Y(I)*CN+X(I)*SN +++
;
        LDA YILO
        LDY YIHI
        JSR MOVFM          ; FAC=Y(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT          ; FAC=Y(I)*CS
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF          ; MEM2=Y(I)*CS
        LDA XILO
        LDY XIHI
        JSR MOVFM          ; FAC=X(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT          ; FAC=X(I)*SN
        LDA #<MEM2
        LDY #>MEM2
        JSR FADD           ; FAC=MEM2+FAC
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF          ; MEM2=FAC
;
; +++ ESEGUE X(I)=MEM1:Y(I)=MEM2 +++
;
        LDA #<MEM1
        LDY #>MEM1
        JSR MOVFM          ; FAC=MEM1
        LDX XILO
        LDY XIHI
        JSR MOVMF          ; X(I)=FAC
        LDA #<MEM2
        LDY #>MEM2
        JSR MOVFM          ; FAC=MEM2
        LDX YILO
        LDY YIHI
        JSR MOVMF          ; Y(I)=FAC
;
; +++ VERIFICA DI FINE CICLO +++
;
        DEC NP
        BEQ EXIT
;
; +++ INCREMENTA PUNTATORI ALLE MATRICI +++
;
        LDA #$05
        CLC
        ADC XILO
        STA XILO
        BCC XNOHI
        INC XIHI
XNOHI
        LDA #$05
        CLC
        ADC YILO
        STA YILO
        BCC YNOHI
        INC YIHI
YNOHI
        JMP INIZIO
;
; +++ RECUPERA REGISTRI DALLO STACK +++
;
EXIT    PLA
        TAY
        PLA
        TAX
        PLA
;
        RTS
END.
```

PROGRAMMA IN BASIC PER LA ROTAZIONE DEL CUBO

```
1000 REM** CUBO RUOTANTE IN BASIC **
1010 IF A=0 THEN A=1: LOAD"PLOTSUB.HEX",8,1
1020 IF A=1 THEN A=2: LOAD"LINESUB.HEX",8,1
1030 REM** DIMENSIONA MATRICI **
1040 NP=8: REM NUMERO DI PUNTI
1050 DIM X(NP), Y(NP), Z(NP)
1060 DIM ED(NP,NP) : REM CONNESSIONI DEI CONTORNI
1070 REM** INIZIALIZZA LE MATRICI **
1080 REM—DATI SULLE COORDINATE DEL CUBO
1090 DATA  75, 75, 75: REM------------------/1
1100 DATA -75, 75, 75: REM    4 PUNTI    /2
1110 DATA -75,-75, 75: REM SUPERIORI /3
1120 DATA  75,-75, 75: REM------------------/4
1130 DATA  75, 75,-75: REM------------------/5
1140 DATA -75, 75,-75: REM    4 PUNTI    /6
1150 DATA -75,-75,-75: REM INFERIORI  /7
1160 DATA  75,-75,-75: REM-----------------/8
1170 REM** RUOTA IL CUBO SU ASSE X DI π/4
1180 FOR I=1 TO NP
1190 READ X(I),Y(I),Z(I)
1200 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1210 Z(I)=Z(I)*COS(π/A)+Y(I)*SIN(π/4)
1220 NEXT
1230 REM** RUOTA IL CUBO SU ASSE Z DI π/4
1240 FOR I=1 TO NP
1250 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1260 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1270 NEXT
1280 REM—DATI SU CONNESSIONI TRA BORDI –
1290 E(1,2)=1: REM 1 CONNESSO 2
1300 E(2,3)=1: E(3,4)=1: E(4,1)=1
1310 E(5,6)=1: REM QUADRATO INFERIORE
1320 E(6,7)=1: E(7,8)=1: E(8,5)=5
1330 E(5,1)=1: REM BORDI VERTICALI
1340 E(6,2)=1: E(7,3)=1: E(8,4)=1
1350 REM** RENDE E(I,J) SIMMETRICA **
1360 FOR I=1 TO NP: FOR J=1 TO NP
1370 IF E(I,J)=1 THEN E(J,I)=1
1380 NEXT j: NEXT I
1390 REM**************************************************
1400 REM**     TRACCIA CUBO RUOTANTE      **
1410 SA=2*π/45
1420 FOR A=π/4 TO π/4+2* π STEP SA
1430 GOSUB 1800: REM RUOTA CON PASSO SA
1440 GOSUB 1590: REM INIZ/PULISCE SCHERMO
1450 REM—TRACCIA IL CUBO –
1460 FOR I=1 TO NP
1470 FOR J=1 TO I
1480 IF E(I,J)=0 THEN 1510: REM NON CONNESSI
1490 GOSUB 1630: REM CALCOLA PROIEZIONE
1500 GOSUB 1670: REM CONGIUNGE PUNTI
1510 NEXT J: NEXT I
1520 REM---------------------------------------------------
1530 NEXT A: REM ANGOLO SUCCESSIVO
1540 REM***********************************************
1550 REM**  ATTENDE  **
1560 GET A$: IF A$="" THEN 1560
1570 GOSUB 1760: REM RESET DELLO SCHERMO
1580 END
1590 REM ** PREDISPONE HI-RES **
1600 POKE 49408,1: POKE 49409,1
1610 POKE 49410,1: SYS 49422
1620 RETURN
1630 REM** CALCOLA PROIEZIONE IN HI-RES **
1640 X1%=X(I)+159: Y1%=199-(Z(I)+100)
1650 X2%=X(J)+159: Y2%=199-(Z(J)+100)
1660 RETURN
1670 REM** LINESUB **
1680 IF(X1%=X2%) AND (Y1%=Y2%) THEN RETURN
1690 MHI=INT(X1%/256): MLO=X1%-256*MHI
1700 NHI=INT(X2%/256): NLO=X2%-256*NHI
1710 POKE 49920,MLO: POKE 49921,MHI
1720 POKE 49922,NLO: POKE 49923,NHI
1730 POKE 49924,Y1%: POKE 49925,Y2%
1740 SYS 49934 : REM LINESUB
1750 RETURN
1760 REM** RESET DELLO SCHERMO **
1770 POKE 49408,0: SYS 49422
1780 PRINT CHR$(147)
1790 RETURN
```

PROGRAMMA DI VERIFICA PER I-ROTSUB

Il seguente programma in BASIC va trascritto e memorizzato sotto il nome "TEST I-ROT". Si noti che nessuna variabile va definita nella sezione che va dalla linea 1880 alla 2010, dove è situata la chiamata SYS 50523, altrimenti gli indirizzi di base delle matrici vengono alterati ed il programma produce risultati imprevedibili.

```
1000 REM** TEST DI I-ROTSUB **
1010 IF A=0 THEN A=1: LOAD"PLOTSUB.HEX",8,1
1020 IF A=1 THEN A=2: LOAD"LINESUB.HEX",8,1
1030 IF A=2 THEN A=3: LOAD"I-ROT.HEX",8,1
1040 REM** DIMENSIONA MATRICI **
1050 NP=8: REM NUMERO DI PUNTI
1060 DIM X(NP), Y(NP), Z(NP)
1070 DIM ED(NP,NP) : REM CONNESSIONI DEI CONTORNI
1080 REM ** INIZIALIZZA LE MATRICI **
1090 REM—DATI SULLE COORDINATE DEL CUBO
1100 DATA  75, 75, 75: REM------------------/1
1110 DATA -75, 75, 75: REM    4 PUNTI    /2
1120 DATA -75,-75, 75: REM SUPERIORI /3
1130 DATA  75,-75, 75: REM------------------/4
1140 DATA  75, 75,-75: REM------------------/5
1150 DATA -75, 75,-75: REM    4 PUNTI    /6
1160 DATA -75,-75,-75: REM INFERIORI  /7
1170 DATA  75,-75,-75: REM-----------------/8
1180 REM** ROTAZIONE ATTORNO ASSE X DI π/4
1190 FOR I=1 TO NP
1200 READ X(I),Y(I),Z(I)
1210 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1220 Z(I)=Z(I)*COS(π/A)+Y(I)*SIN(π/4)
1230 NEXT
1240 REM** ROTAZIONE ATTORNO ASSE Z DI π/4
1250 FOR I=1 TO NP
1260 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1270 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1280 NEXT
1290 REM—DATI SU CONNESSIONI TRA BORDI –
1300 E(1,2)=1: REM 1 CONNESSO 2
1310 E(2,3)=1: E(3,4)=1: E(4,1)=1
1320 E(5,6)=1: REM QUADRATO INFERIORE
1330 E(6,7)=1: E(7,8)=1: E(8,5)=5
1340 E(5,1)=1: REM BORDI VERTICALI
1350 E(6,2)=1: E(7,3)=1: E(8,4)=1
1360 REM** RENDE E(I,J) SIMMETRICA **
1370 FOR I=1 TO NP: FOR J=1 TO NP
1380 IF E(I,J)=1 THEN E(J,I)=1
1390 NEXT j: NEXT I
1400 REM**************************************************
1410 REM**     TRACCIA CUBO RUOTANTE      **
1420 SA=2*π/45: CS=COS(SA): SN=SIN(SA)
1430 FOR A=0 TO 2*π STEP SA
1440 GOSUB 1870: REM RUOTA CON PASSO SA
1450 GOSUB 1600: REM INIZ/PULISCE SCHERMO
1460 REM—TRACCIA IL CUBO –
1470 FOR I=1 TO NP
1480 FOR J=1 TO I
1490 IF E(I,J)=0 THEN 1520: REM NON CONNESSI
1500 GOSUB 1640: REM CALCOLA PROIEZIONE
1510 GOSUB 1680: REM CONGIUNGE PUNTI
1520 NEXT J: NEXT I
1530 REM---------------------------------------------------
1540 NEXT A: REM ANGOLO SUCCESSIVO
1550 REM***********************************************
1560 REM**  ATTENDE  **
1570 GET A$: IF A$="" THEN 1570
1580 GOSUB 1770: REM RESET DELLO SCHERMO
1590 END
1600 REM ** PREDISPONE HI-RES **
1610 POKE 49408,1: POKE 49409,1
1620 POKE 49410,1: SYS 49422
1630 RETURN
1640 REM** CALCOLA PROIEZIONE IN HI-RES **
1650 X1%=X(I)+159: Y1%=199-(Z(I)+100)
1660 X2%=X(J)+159: Y2%=199-(Z(J)+100)
1670 RETURN
1680 REM** LINESUB **
1690 IF(X1%=X2%) AND (Y1%=Y2%) THEN RETURN
1700 MHI=INT(X1%/256): MLO=X1%-256*MHI
1710 NHI=INT(X2%/256): NLO=X2%-256*NHI
1720 POKE 49920,MLO: POKE 49921,MHI
1730 POKE 49922,NLO: POKE 49923,NHI
```

```
1800 REM** RUOTA IL CUBO SU ASSE Z DI SA **
1810 FOR I=1 TO NP
1820 X(I)=X(I)*COS(SA)-Y(I)*SIN(SA)
1830 Y(I)=Y(I)*COS(SA)+X(I)*SIN(SA)
1840 NEXT
1850 RETURN
```

```
1740 POKE 49924,Y1%: POKE 49925,Y2%
1750 SYS 49934 : REM LINESUB
1760 RETURN
1770 REM** RESET DELLO SCHERMO **
1780 POKE 49408,0: SYS 49422
1790 PRINT CHR$(147)
1800 RETURN
1810 REM** RUOTA IL CUBO SU ASSE Z DI SA **
1820 FOR I=1 TO NP
1830 X(I)=X(I)*CS-Y(I)*SN
1840 Y(I)=Y(I)*CS+X(I)*SN
1850 NEXT
1860 RETURN
1870 REM** RUOTA ATTORNO ASSE Z DI SA
1880 X(1)=X(1): REM X(1) VAR CORRENTE
1890 POKE 50500,PEEK(71). REM X(1) LO
1900 POKE 50501,PEEK(72): REM X(1) HI
1910Y(1)=Y(1): REM Y(1) VAR CORRENTE
1920 POKE 50502,PEEK(71) : REM Y(1) LO
1930 POKE 50503,PEEK(72): REM Y(1) HI
1940 POKE 50504,NP: REM NUMERO DI PUNTI
1950 CS=CS: REM CS VAR CORRENTE
1960 POKE 50509,PEEK(71)
1970 POKE 50510,PEEK(72)
1980 SN=SN: REM SN VAR CORRENTE
1990 POKE 50511,PEEK(71)
2000 POKE 50512,PEEK(72)
2010 SYS 50523
2020 RETURN
```

Rivoluzione grafica

La seconda ed ultima parte dedicata alla grafica 3D

Il programma ibrido (*Test I-Rot* e *I-Rot.Hex*) presentato nel capitolo precedente, funziona abbastanza velocemente. Tuttavia la scansione della matrice e%(i,j), necessaria per individuare i punti che devono essere tracciati, rallenta l'esecuzione del programma. La matrice E%(), si ricorderà, definisce i nodi collegati tra loro da linee visibili. Per eliminare questo inconveniente, occorre programmare in codice macchina il resto del ciclo BASIC del programma *Cubo Rotante* e mandare questo in esecuzione. Solo così si ottiene un aumento di velocità. Per poter eseguire i seguenti calcoli:

```
X1%=X(I)+159: Y1%=199-(Z(I)+100)
X2%=X(J)+159: Y2%=199-(Z(J)+100)
```

originariamente inseriti nelle righe 1650 e 1660 del programma BASIC, è necessario chiamare delle nuove routine dell'interprete. In concreto: entrambe le linee sommano, alla variabile X(I) in virgola mobile, il valore 159 (anch'esso in virgola mobile) prima di prenderne la parte intera ed immagazzinarla come variabile intera X1% in 2 byte. Ecco le routine dell'interprete che fanno il lavoro:

- FLPINT (indirizzo $B1AA):
  Questa routine inserisce in FAC la parte intera del numero, calcola il risultato (se compreso tra i valori –32767 e +32767) nel formato byte-basso/byte-alto e lo trascrive rispettivamente nei registri Y ed A. Si noti l'insolita disposizione basso/alto, opposta a quella della maggior parte delle routine dell'interprete.
- SNGFT (indirizzo $B3A2):
  Questa routine estrae dal registro Y un intero a singolo byte (compreso tra 0 e 255), e lo inserisce in FAC nel formato in virgola mobile. SNGFT viene usata dalla subroutine SETUP del listato Assembly (linea 5150) che inserisce il valore 159 nel registro Y e chiama la routine SNGFT per trasformare il numero ed inserirlo in FAC. Quindi usa MOVMF per copiare il risultato nei 5 byte di MEM1, dove viene conservato e, successivamente, utilizzato quando occorre sommare 159.

Gli altri problemi, relativi alla conversione in codice macchina del ciclo BASIC, riguardano il caricamento degli elementi della matrice che definisce la forma della figura da ruotare. Infatti, in alcuni casi, il calcolo dei puntatori alla matrice può risultare difficoltoso. D'altra parte le matrici X(I), Y(I), Z(I) delle coordinate, non creano particolari problemi poiché basta aggiungere 5 byte, al puntatore in ciascuna matrice, per ottenere l'indirizzo dell'elemento successivo.

La matrice bidimensionale e%(I,J) pone problemi diversi. I suoi elementi vengono registrati in memoria in modo sequenziale:

E%(0,0),E%(1,0),E%(2,0)... E%(NP,0)
E%(0,1),E%(1,1),E%(2,1)... E%(NP,1) e così via...

Quindi la matrice si compone di blocchi di memoria, ciascuno lungo 2x(NP+1): ogni blocco corrisponde ai valori del secondo indice e ciascun elemento occupa due byte (perché la matrice contiene valori interi).
Il nostro obiettivo è “tradurre” nel modo più esatto il programma BASIC nel codice macchina in modo che i cicli I,J, che esaminano la matrice E%(I,J), presentino il formato:

FOR I=1 TO NP
FOR j=1 TO NP

Per ottenere un codice macchina equivalente al NEXT I, questo complica notevolmente la modifica da apportare al puntatore, considerando che vanno saltati tutti gli elementi il cui primo indice è 0. Per ottenere la rotazione della figura, gli elementi di E%(I,J) vanno elaborati nel seguente ordine:

E%(1,1)
E%(1,2) E%(2,2)
E%(1,3) E%(2,3) E%(3,3)
E%(1,4) E%(2,4) E%(3,4) E%(4,4) ecc.

Un rapido calcolo dà l’espressione 2x(NP+1), ossia il valore da sommare al puntatore ogni volta che viene incrementato I. Con il codice macchina del 6510, la tecnica più conveniente per accedere agli elementi di E%(I,J) è quella dell’indirizzamento indiretto che viene programmato con le istruzioni:

```
LDY JINDEX
LDA (ZPTEMP),Y
```

ZPTEMP è un puntatore a due byte situato in pagina zero, mentre JINDEX viene usato per registrare i valori di J. ZPTEMP va incrementato dopo ogni aumento di J. Dall’incremento, applicato a ZPTEMP ed al registro Y, dipende l’aumento di due byte dell’offset necessario per ogni successivo incremento di J. Quindi, come risultato finale di queste complesse operazioni, adogni iterazione nel ciclo di I ZPTEMP viene incrementato di (2xNP+1)-(I-1). Il valore (I-1) viene sottratto dalla lunghezza del blocco in quanto ZPTEMP è stato incrementato (I-1)(volte dal cilco di J appena concluso. L’esecuzione di quest espressione, per il calcolo dell’offset, porta il vettore ZPTEMP a puntare al byte giusto dopo che I è stato incrementato.
Un’ultima coinsiderazione: sarebbe assai conveniente poter chiamare una routine dell’interprete capace di localizzare direttamente la variabile E%(I,J). Tale routine esiste ma, sfortunatamente, il suo uso è molto complesso (in effetti funziona con tutti i possibili tipi di variabile) e l’esecuzione lenta. L’offset, pertanto, va calcolato nel modo appena descritto, a partire dall’indirizzo di E%(1,1).

ROUTINE DI ROTAZIONE

Il seguente listato è relativo al programma in codice macchina che fa ruotare la figura geometrica definita dalla matrice E%(I,J). Il programma sfrutta le routine dell’interprete e le tecniche di individuazione delle variabili descritte nel testo. Il primo obiettivo (convertire in linguaggio macchina il ciclo BASIC che legge le matrici, per visualizzare una figura in rotazione) è stato, dunque, raggiunto. Inoltre vengono forniti un programma di caricamento in BASIC del codice macchina ed un programma BASIC di prova che prepara le variabili per il codice macchina e chiama la routine. Questo programma può essere utilizzato con una versione assemblata del codice sorgente (“II-ROT.HEX”) o, in alternativa, caricando ed eseguendo il programma di caricamento facendolo seguire dal comando NEW ed infine caricando e lanciando il programma di prova. Nel secondo caso si elimini la linea 1030 dal programma di prova.

CARICATORE IN BASIC

```
1000 REM** INSERISCE LA VERSIONE II-ROT.HEX **
1010 REM***************************************************
1020 DATA 72,138,72,152,72,32,101,199
1030 DATA 173,83,197,172,84,197,32,162
1040 DATA 187,173,75,197,172,76,197,32
1050 DATA 40,186,162,94,160,197,32,212
1060 DATA 187,173,87,197,172,88,197,32
1070 DATA 162,187,173,77,197,172,78,197
1080 DATA 32,40,186,169,94,160,197,32,80
1090 DATA 184,162,94,160,197,32,212,187
1100 DATA 173,87,197,172,88,197,32,162
1110 DATA 187,173,75,197,172,76,197,32
1120 DATA 40,186,162,99,160,197,32,212
1130 DATA 187,173,83,197,172,84,197,32
1140 DATA 162,187,173,77,197,172,78,197
```

LISTATO ASSEMBLY

```
1010 ; ++++++++++++++++++++
1020 ; ++       ROTSUB 64       ++
1030 ; ++                        ++
1040 ; ++++++++++++++++++++
1140 ;
1150 *                    = $C544
1190 ; VARIABILI CHIAMATE DAL BASIC
1200 ;
1210 XBASLO       *=*+1      ; POKE 50500,X(1) LO
1220 XBASHI       *=*+1      ; POKE 50501,X(1) HI
1230 XBASLO       *=*+1      ; POKE 50502,Y(1) LO
1240 XBASHI       *=*+1      ; POKE 50503,Y(1) HI
1250 XBASLO       *=*+1      ; POKE 50504,Z(1) LO
1260 XBASHI       *=*+1      ; POKE 50505,Z(1) HI
1270 NP           *=*+1      ; POKE 50506,NP
1280 CSLO         *=*+1      ; POKE 50507,CSLO
```

```
1150 DATA 32,40,186,169,99,160,197,32
1160 DATA 103,184,162,99,160,197,32,212
1170 DATA 187,169,94,160,197,32,162,187
1180 DATA 174,83,197,172,84,197,32,212
1190 DATA 187,169,99,160,197,32,162,187
1200 DATA 174,87,197,172,88,197,32,212
1210 DATA 187,206,93,197,240,31,169,5,24
1220 DATA 109,83,197,141,83,197,144,3
1230 DATA 238,84,197,169,5,24,109,87,197
1240 DATA 141,87,197,144,3,238,88,197,76
1250 DATA 112,197,169,1,141,0,193,141,1
1260 DATA 193,141,2,193,32,14,193,32,101
1270 DATA 199,169,1,141,81,197,141,82
1280 DATA 197,173,79,197,133,253,173,80
1290 DATA 197,133,254,172,82,197,177,253
1300 DATA 208,3,76,207,198,173,83,197
1310 DATA 172,84,197,32,162,187,169,94
1320 DATA 160,197,32,103,184,32,170,177
1330 DATA 140,0,195,141,1,195,173,85,197
1340 DATA 172,86,197,32,162,187,169,94
1350 DATA 160,197,32,103,184,32,170,177
1360 DATA 140,2,195,141,3,195,173,89,197
1370 DATA 172,90,197,32,162,187,169,99
1380 DATA 160,197,32,80,184,32,170,177
1390 DATA 140,4,195,173,91,197,172,92
1400 DATA 197,32,162,187,169,99,160,197
1410 DATA 32,80,184,32,170,177,140,5,195
1420 DATA 173,0,195,205,2,195,208,19,173
1430 DATA 1,195,205,3,195,208,11,173,4
1440 DATA 195,205,5,195,208,3,76,207,198
1450 DATA 32,14,195,173,81,197,205,82
1460 DATA 197,240,40,169,5,24,109,85,197
1470 DATA 141,85,197,144,3,238,86,197
1480 DATA 169,5,24,109,91,197,141,91,197
1490 DATA 144,3,238,92,197,230,253,208,2
1500 DATA 230,254,238,82,197,76,73,198
1510 DATA 173,74,197,205,81,197,240,88
1520 DATA 169,5,24,109,83,197,141,83,197
1530 DATA 144,3,238,84,197,169,5,24,109
1540 DATA 89,197,141,89,197,144,3,238,90
1560 DATA 237,81,197,24,105,1,24,101,253
1570 DATA 133,253,165,254,105,0,133,254
1580 DATA 173,68,197,141,85,197,173,69
1590 DATA 197,141,86,197,173,72,197,141
1600 DATA 91,197,173,73,197,141,92,197
1610 DATA 169,1,141,82,197,238,81,197,76
1620 DATA 73,198,104,168,104,170,104,96
1630 DATA 173,68,197,141,83,197,141,85
1640 DATA 197,173,69,197,141,84,197,141
1650 DATA 86,197,173,70,197,141,87,197
1660 DATA 173,71,197,141,88,197,173,74
1670 DATA 197,141,93,197,173,72,197,141
1680 DATA 89,197,141,91,197,173,73,197
1690 DATA 141,90,197,141,92,197,160,159
1700 DATA 32,162,179,162,94,160,197,32
1710 DATA 212,187,160,99,32,162,179,162
1720 DATA 99,160,197,32,212,187,96
1730 DATA 79160: REM* CHECKSUM *
1740 FOR I=50536 TO 51123
1750 READ X: POKE I,X: CC=CC+X
1760 NEXT
1770 READ X: IL X<>CC THEN PRINT "ERRORE CHECKSUM":
          STOP
1780 PRINT "II – ROT.HEX CARICAMENTO OK"
```

```
1290 CSHI        *=*+1      ; POKE 50508,CSHI
1300 SNLO        *=*+1      ; POKE 50509,SNLO
1310 SNHI        *=*+1      ; POKE 50510,SNHI
1320 EBASLO      *=*+1      ; POKE 50511,E%(1,1) LO
1330 EBASHI      *=*+1      ; POKE 50512,e%(1,1) HI
1340 ;
1350 ; VARIABILI USATE DAL CODICE MACCHINA
1370 IINDEX      *=*+1
1380 JNDEX       *=*+1
1390 XILO        *=*+1
1400 XIHI        *=*+1
1410 XJLO        *=*+1
1420 XJHI        *=*+1
1430 YILO        *=*+1
1440 YIHI        *=*+1
1450 ZILO        *=*+1
1460 ZIHI        *=*+1
1470 ZJLO        *=*+1
1480 ZJHI        *=*+1
1490 TEMPNP      *=*+1
1500 MEM1        *=*+1                ; variabile in virgola mobile
1510 MEM2        *=*+1                ; variabile in virgola mobile
1520 ZTEMP        =$FD                ; locazione libera di pag. zero
1530 ;
1540 ; CHIAMATE ARITMETICHE DELL'INTERPRETE
1560 FMULT        =$BA28              ; FAC=FAC*MEM
1570 FSUB         =$B850              ; FAC=MEM-FAC
1580 FADD         =$B867              ; FAC=FAC+MEM
1590 MAVFM        =$BBA2              ; FAC=MEM
1600 MOVMF        =$BBD4              ; MEM=FAC
1610 FLPINT       =$B1AA              ; .Y/.A=INT(FAC)
                                        N.B. formato basso/alto !
1620 SNGFT        =$B3A2              ; FAC=.Y
1630 ;
1640 ; ALTRE ROUTINE PER VARIABILI IN C/M
1660 LINSUB       =$C30E
1670 MLO          =$C300
1680 MHI =$C301
1690 NLO          =$C302
1700 NHI          =$C303
1710 Y1           =$C304
1720 Y2           =$C305
1740 ; +++ SALVA I REGISTRI +++
1760             PHA
1770             TXA
1780             PHA
1790             TYA
1800             PHA
1820 ; +++ INIZIALIZZA LE VARIABILI +++
1840             JSR SETUP
1860 ; +++ CALCOLA MEM1=X(I)*CS-Y(I)*SN +++
1880 INIZIO
1890             LDA XILO
1900             LDY XIHI
1910             JSR MOVFM          ; FAC=X/I)
1920             LDA CSLO
1930             LDY CSHI
1940             JSR FMULT          ; FAC=X(I)*CS
1950             LDX #<MEM1
1960             LDY #>MEM1
1970             JSR MOVMF          ; MEM1=X(I)*CS
1980             LDA XILO
1990             LDY YIHI
2000             JSR MOVFM          ; FAC=Y(I)
2010             LDA SNLO
2020             LDY SNHI
2030             JSR FMULT          ; FAC=Y(I)*SN
2040             LDA #<MEM1
2050             LDY #>MEM1
2060             JSR FSUB           ; FAC=MEM1-FAC
2070             LDX #<MEM1
2080             LDY #>MEM1
2090             JSR MOVFM          ; MEM1=FAC
2110 ; +++ CALCOLA MEM2=Y(I)*CS+X(1)*SN +++
2130             LDA YILO
2140             LDY YIHI
2150             JSR MOVFM          ; FAC=Y(I)
2160             LDA CSLO
2170             LDY CSHI
2180             JSR FMULT          ; FAC=Y(I)*CS
2190             LDX #<MEM2
2200             LDY #>MEM2
2210             JSR MOVMF          ; MEM2=Y(I)*CS
```

PROGRAMMA DI PROVA BASIC

```
1000 REM** TRACCIA I CUBI IN ROTAZIONE **
1010 IF A=0 THEN A=1: LOAD"PLOTSUB.HEX",8,1
1020 IF A=1 THEN A=2: LOAD"LINESUB.HEX",8,1
1030 IF A=2 THEN A=3: LOAD"II –ROT.HEX",8,1
1040 PRINT CHR$(147) "ATTENDI 17 SECONDI"
1050 REM** DIMENSIONA LE MATRICI **
1060 NP=24: REM NUMERO DEI PUNTI
1070 DIM X(NP), Y(NP), Z(NP)
1080 DIM E%(NP,NP) : REM COLLEGAMENTI DEGLI SPIGOLI
1090 REM** INIZIALIZZA LE MATRICI **
1100 REM—COORDINATE INIZIALI DEL CUBO
1110 DATA  50, 50, 50: REM -----------------  /1
1120 DATA -50, 50, 50: REM I 4 PUNTI      /2
1130 DATA -50,-50, 50: REM SUPERIORI  /3
1140 DATA  50,-50, 50: REM -----------------  /4
1150 DATA  50, 50,-50: REM -----------------  /5
1160 DATA -50, 50,-50: REM I 4 PUNTI      /6
1170 DATA -50,-50,-50: REM INFERIORI   /7
1180 DATA   50,-50-50: REM -----------------  /8
```

La sezione successiva del programma carica i dati, appena definiti, nelle matrici X(), Y() e Z(). Si noti che, per cubi piccoli, i valori iniziali vengono ridotti di un fattore 0,3.

```
1190 REM** CARICA I DATI PER 3 CUBI **
1200 FOR I=1 TO 8: REM CUBO GRANDE CENTRALE
1210 READ X(I),Y(I),Z(I)
1220 NEXT
1230 RESTORE
1240 FOR I=9 TO 16: REM CUBO PICCOLO DI DESTRA
1250 READ X,Y,Z
1260 X(I)=.3*X+120: Y(I)=.3*Y: Z(I)=.3*Z
1270 NEXT
1280 RESTORE
1290 FOR I=7 TO 24: REM CUBO PICCOLO DI SINISTRA
1300 READ X,Y,Z
1310 X(I)=.3*X-120: Y(I)=.3*Y: Z(I)=.3*Z
1320 NEXT
```

La sezione seguente ruota di 45º verso destra le coordinate dei punti per i tre cubi.

```
1330 REM** RUOTA LE FIGURE DI π/4 SULL'ASSE X **
1340 FOR I=1 TO NP
1350 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1360 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1370 NEXT
1380 REM** RUOTA LE FIGURE DI π/4 SULL'ASSE Z **
1390 FOR I=1 TO NP
1400 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1410 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1420 NEXT
```

La prossina sezione del programma carica, nella matrice delle connessioni E%(), i punti che devono essere collegati all'interno della figura per ciascuno dei tre cubi.

```
1430 REM** COLLEGAMENTO DEGLI SPIGOLI **
1440 REM—CUBO GRANDE CENTRALE --
1450 E%(1,2)=1: REM 1 COLLEGATO 2
1460 E%(2,3)=1: E%(3,4)=1: E%(4,1)=1
1470 E%(5,6)=1: REM QUADRATO INFERIORE
1480 E%(6,7)=1: E%(7,8)=1: E%(8,5)=1
1490 E%(5,1)=1: REM SPIGOLI LATERALI
1500 E%(6,2)=1: E%(7,3)=1: E%(8,4)=1
1510 REM--------------------------------------
1520 REM ** CUBO PICCOLO DI DESTRA **
1530 FOR I=9 TO 16: FOR J= 9 TO 16
1540 E%(I,J)=E%(I-8,J-8)
1550 NEXT: NEXT
1560 REM ** CUBO PICCOLO DI SINISTRA **
1570 FOR I=17 TO 24: FOR J=17 TO 24
1580 E%(I,J)=E%(I-8,J-8)
1590 NEXT: NEXT
1600 REM ** PONE IN SIMMETRIA E%(I,J) **
1610 FOR I=1 TO NP: FOR J=1 TO NP
1620 IF E%(I,J)<>0 THEN E%(J,I)=1
```

```
2220              LDA XILO
2230              LDY XIHI
2240              JSR MOVFM        ; FAC=X(I)
2250              LDA SNLO
2260              LDY SNHI
2270              JSR FMULT        ; FAC=X(I)*SN
2280              LDA #<MEM2
2290              LDY #>MEM
2300              JSR FADD         ; FAC=MEM2+FAC2
2310              LDX #<MEM2
2320              LDY #>MEM2
2330              JSR MOVMF        ; MEM2=FAC
2350 ; +++ CALCOLA C(I)=MEM1:Y(I)=MEM2 +++
2370              LDA #<MEM1
2380              LDY #>MEM1
2390              JSR MOVFM        ; FAC=MEM1
2400              LDX XILO
2410              LDY XIHI
2420              JSR MOVMF        ; X(I)=FAC
2430              LDA #<MEM2
2440              LDY #>MEM2
2450              JSR MOVFM        ; FAC=MEM2
2460              LDX YILO
2470              LDY YIHI
2480              JSR MOVMF        ; Y(I)=FAC
2500 ; +++ VERIFICA DI FINE CICLO +++
2520              DEC TEMPNP
2530              BEQ CONTIN
2550 ; +++ INCREMENTA I PUNTATORI ALLA MATRICE +++
2570              LDA #$05
2580              CLC
2590              ADC XILO
2600              STA XILO
2610              BCC XNOHI
2620              INC XIHI
2630 XNOHI
2640              LDA #$05
2650              CLC
2660              ADC YILO
2670              STA YILO
2680              BCC YNOHI
2690              INC YIHI
2700 YNOHI
2710              JMP INIZIO
2730 ; +++ CANCELLA/INIZIALIZZA LO SCHERMO +++
2750 CONTIN
2760              LDA #$01
2770              STA $C100
2780              STA $C101
2790              STA $C102
2800              JSR $C10E        ; seleziona l'alta risoluzione
2820 ; +++ TRACCIA LE LINEE DA E%(I,J) +++
2840 ; INIZIALIZZA LE VARIABILI
2860              JSR SETUP
2870              LDA #$01
2880              STA IINDEX       ; I=1
2890              STA JINDEX       ; J=1
2900              LDA EBASLO       ; memorizza il
2910              STA ZPTEMP       ; puntatore E%
2920              LDA EBASHI       ; in pagina
2930              STA ZPTEMP+1     ; zero
2950 ; USCITA-INIZIA IL CICLO GIGANTE I,J
2970 SUCCIJ
2980              LDY JINDEX
2990              LDA (ZPTEMP),Y   ; get E%(I,J)
3000              BNE ESEGUE
3010              JMP ONWARD
3030 ; +++ CALCOLA X1%=X(I)+159 +++
3040 ;       MLO=X1% LO:MHI=X1% HI
3060 ESEGUE
3070              LDA XILO
3080              LDY XIHI
3090              JSR MOVFM        ; FAC=X(I)
3100              LDA #<MEM1
3110              LDY #>MEM1
3120              JSR FADD         ; FAC=X(I)+159
3130              JSR FLPINT       : .Y/.A=INT(FAC)
3140              STY MLO          ; X1% LO
3150              STA MHI          X1% HI
3170 ; +++ CALCOLA X2%=X(J)+159 +++
3180 ;
3200              LDA XJLO
3210              LDY XJHI
```

```
1630 NEXT: NEXT
```

Infine il programma chiama il codice macchina che esegue i calcoli di rotazione necessari e traccia la nuova immagine. Questa chiamata viene ripetuta più volte da un ciclo che ruota i tre cubi di 360° prima di terminare.

```
1640 REM******************************************
1650 REM** TRACCIA IL CUBO RUOTATO **
1660 SA=2*π/45: CS=COS(SA): SN=SIN(SA)
1670 GOSUB 1790 : REM INIZIALIZZA
1680 FOR A=0 TO 2+π STEP SA
1690 SYS 50536: REM ESEGUE LA ROTAZIONE
1700 NEXT A: REM ANGOLO SUCCESSIVO
1710 GET A$: IL A$="" THE 1710
1720 REM******************************************
1730 GOSUB 1750: REM CANCELLA LO SCHERMO
1740 END
1750 REM** CANCELLA LO SCHERMO **
1760 POKE 49408,0: SYS 49422
1770 PRINT CHR$(147)
1780 RETURN
1790 REM** INIZIALIZZA ROTSUB **
1800 REM****************************************************
1810 REM** N.B. NON DEFINIRE NUOVE VARIABILI **
1820 REM**      TRA QUESTE SUBROUTINE E LA     **
1830 REM**      CHIAMATA DELLA ROTAZIONE IN    **
1840 REM**             CODICE MACCHINA.         **
1850 REM**  POTREBBE CAMBIARE GLI INDIRIZZI  **
1860 REM**          DI BASE DELLE MATRICI         **
1870 REM****************************************************
1880 X(1)=X(1): REM X(1) VAR. CORRENTE
1890 POKE 50500,PEEK(71): REM X(1) LO
1900 POKE 50501,PEEK(72): REM X(1) HI
1910 Y(1)=Y(1): REM Y(1) VAR. CORRENTE
1920 POKE 50502,PEEK(71): REM Y(1) LO
1930 POKE 50503,PEEK(72): REM Y(1) HI
1940 Z(1)=Z(1): REM VAR. CORRENTE
1950 POKE 50504,PEEK(71)
1960 POKE 50505,PEEK(72)
1970 POKE 50506,NP: REM NUMERO DI PUNTI
1980 CS=CS: REM CS= VAR. CORRENTE
1990 POKE 50507,PEEK(71)
2000 POKE 50508,PEEK(72)
2010 SN=SN: REM SN= VAR. CORRENTE
2020 POKE 50509,PEEK(71)
2030 POKE 50510,PEEK(72)
2040 E%(1,1)=E%(1,1): REM E% = VAR. CORRENTE
2050 POKE 50511,PEEK(71)
2060 POKE 50512,PEEK(72)
2070 RETURN
```

```
3220          JSR MOVFM       ; FAC=X(J)
3230          LDA #<MEM1
3240          LDY #>MEM1
3250          JSR FADD        ; FAC=X(J)+159
3260          JSR FLPINT      ;.Y/.A=INT(FAC)
3270          STY NLO         ; X2% LO
3280          STA NHI         ; X2% HI
3300 ; +++ CALCOLA Y1%=99-Z(I) +++
3320          LDA ZILO
3330          LDY ZIHI
3340          JSR MOVFM       ; FAC=Z(I)
3350          LDA #<MEM2
3360          LDY #>MEM2
3370          JSR FSUB        ; FAC=99-Z(1)
3380          JSR FLPINT      ; .Y/.A=INT(FAC)
3390          STY Y1          ; Y1%
3410 ; +++ CALCOLA Y2%=99-Z(J) +++
3430          LDA ZJLO
3440          LDY ZJHI
3450          JSR MOVFM       ; FAC=Z(J)
3460          LDA #<MEM2
3470          LDY #>MEM2
3480          JSR FSUB        ; FAC=99-Z(J)
3490          JSR FLPINT      ; .Y/.A=INT(FAC)
3500          STY Y2
3520 ; PRONTO PER LINSUB
3540 ; CONFRONTO A DUE BYTE PER X1%=X2%
3560          LDA MLO
3570          CMP NLO
3580          BNE NOPE
3590          LDA MHI
3600          CMP NHI
3610          BNE NOPE
3630 ; CONFRONTA Y1=Y2
3650          LDA Y1
3660          CMP Y2
3670          BNE NOPE
3690 ; TUTTI UGUALI QUINDI EVITA LINSUB
3710          JMP ONWARD
3720 NOPE
3730          JSR LINSUB      ; traccia una linea
4550 ; 3750 ; INCREMENTA I PUNTATORI A J
3770 ONWARD
3780          LDA IINDEX      ; prima verifica
3790          CMP JINDEX      ; J=<i?
3800          BEQ NEXTI
3820 ; INCREMENTA XJLO/XJHI
3840          LDA #$05
3850          CLC
3860          ADC XJLO
3870          STA XJLO
3880          BCC XJNOHI
3890          INC XJHI
3900 XJNOHI
3920 ; INCREMENTA ZILO/ZJHI
3940          LDA #$05
3950          CLC
3960          ADC ZJLO
3970          BTA ZJLO
3980          BCC ZJNOHI
3990          INC ZJHI
4000 ZJNOHI
4020 ; INCREMENTA DI UNO IL PUNTATORE DI E%
4040          INC ZPTEMP
4050          BNE NOHIBY
4060          INC ZPTEMP+1
4070 NOHIBY
4080          INC JINDEX
4090          JMP NEXTIJ
4110 ; INCREMENTA I PUNTATORI
4130 SUCCI
4140          LDA NP          ; prima verifica
4150          CMP IINDEX      ; I=<NP?
4180 ; INCREMENTA XILO/XIHI
4200          LDA #$05
4210          CLC
4220          ADC XILO
4230          STA XILO
4240          BCC XINOHI
4250          INC XIHI
4260 XINOHI
4280 ; INCREMENTA ZILO/ZIHI
4300          LDA #$05
```

```
4310            CLC
4320            ADC ZILO
4330            STA ZILO
4340            BCC ZINOHI
4350            INC ZIHI
4360 ZINOHI
4380 ; INCREMENTA I PUNTATORI A E%
4400            LDA #$04        ; quando incrementa I
4410            CLC             ; va sommato
4420            ADC NP          ; 2*(NP+1)-(I-1)
4430            ASL A           ; al puntatore
4440            SEC             ; ZP per
4450            SBC IINDEX      ; E%
4460            CLC
4470            ADC #$01
4480            CLC
4490            ADC ZPTEMP
4500            STA ZPTEMP
4510            LDA ZPTEMP+1
4520            ADC #$00
4530            STA ZPTEMP+1
REINIZIALIZZA XJLO/XJHI
4570            LDA XBASLO
4580            STA XJLO
4590            LDA XBASHI
4600            STA XJHI
4620 ; REINIZIALIZZA ZJLO/ZJHI
4640            LDA ZBASLO
4650            STA ZJLO
4660            LDA ZBASHI
4670            STA ZJHI
4690 ; REINIZIALIZZA JINDEX
4710            LDA #$01
4720            STA JINDEX
4740 ; INCREMENTA IINDEX
4760            INC IINDEX
4770            JMP NEXTIJ
4790 ; TERMINA IL CICLO GIGANTE
4800 ;
4810 ; +++ ESTRAE I VALORI DAI REGISTRI +++
4820 ;
4830 USCITA
4840            PLA
4850            TAY
4860            PLA
4870            TAX
4880            PLA
4890            RTS
4930 ; +++ SUBROUTINE ++
4940 ;
4950 CREA
4960            LDA ZBASLO
4970            STA XILO
4980            STA XJLO
4990            LDA ZBASHI
5000            STA XIHI
5010            STA XJHI
5020            LDA YBASLO
5030            STA YILO
5040            LDA YBASHI
5050            STA YIHI
5060            LDA NP
5070            STA TEMPNP
5080            LDA ZBASLO
5090            STA ZILO
5100            STA ZJLO
5110            LDA ZBASHI
5120            STA ZIHI
5130            STA ZJHI
5140            LDY #159
5150            JSR SNGFT       ; FAC=159
5160            LDX #<MEM1
5170            LDY #>MEM1
5180            JSR MOVMF       ; MEM1=159 IN FPT
5190            LDY #99
5200            JSR SNGFT       ; FAC=99
5210            LDX #<MEM2
5220            LDY #>MEM2
5230            JSR MOVMF       ; MEM2=99 IN FTP
5240            RTS
```

Un'interfaccia MIDI

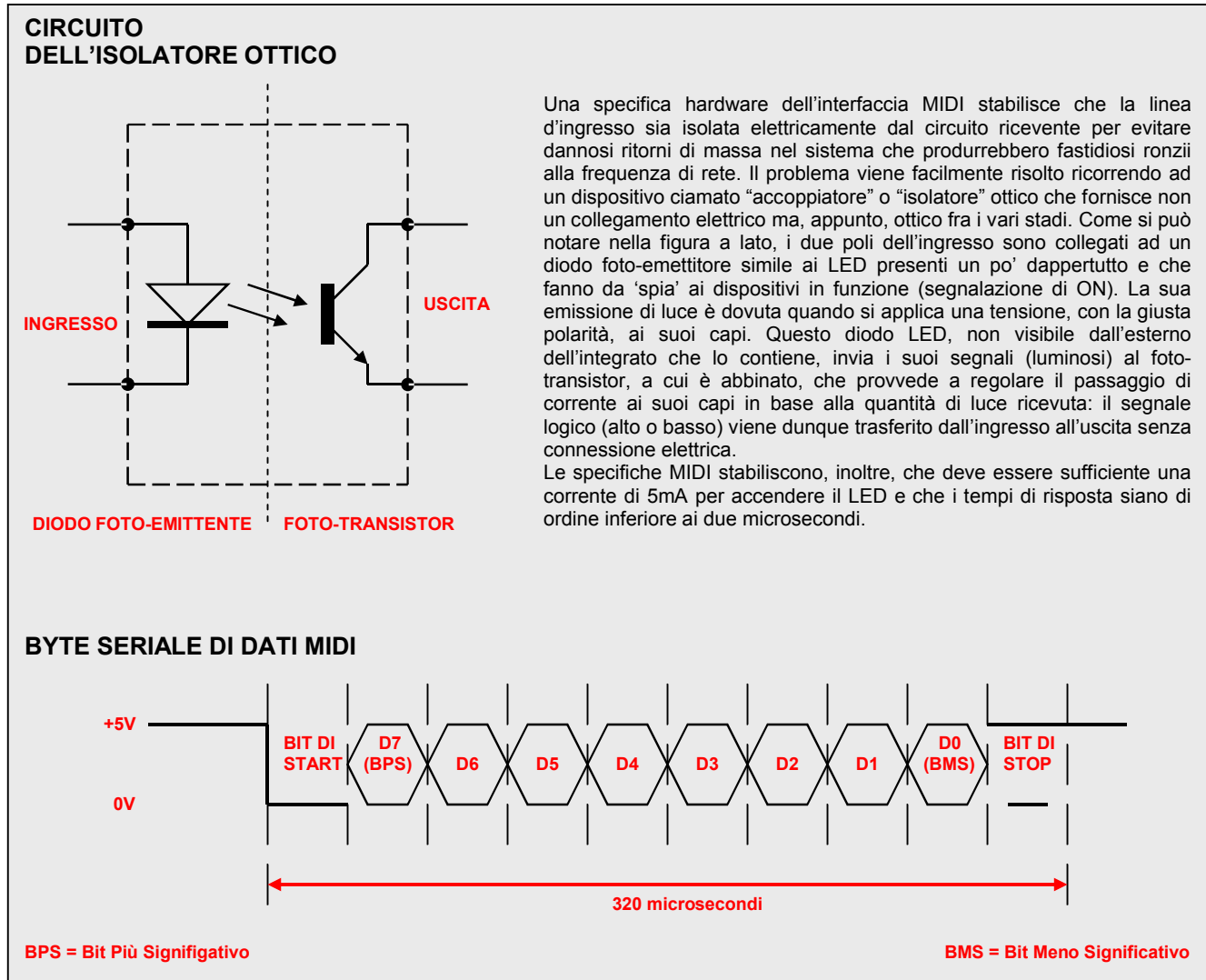
Un progetto per comunicare con strumenti musicali

Le specifiche della Musical Instrument Digital Interface (MIDI) forniscono indicazioni generali circa l'hardware ed il software necessario per creare un sistema nel quale, gli strumenti musicali digitali, comunicano tra loro tramite l'interfaccia MIDI. Poiché le informazioni sono di tipo digitale, non vi sono difficoltà nell'introdurre, nel sistema, anche un computer in modo da ottenere un controllo altamente organizzato e coordinato dei vari strumenti. Oltre tutto questo consentirà di conservare su nastro o disco le sequenze digitali dei codici di controllo. La sezione hardware dell'interfaccia MIDI, qui presentata, ha un costo veramente accessibile e, l'intera realizzazione, è di estrema facilità per chi abbia un minimo di esperienza con il saldatore.
Una volta realizzata l'interfaccia, è sufficiente scrivere un programma che consenta di pilotare il sistema usando semplici comandi e parametri (il listato di un simile programma viene fornito nel capitolo successivo).
Da quando è stato introdotto l'uso dell'interfaccia MIDI, nell'agosto del 1983, il settore della musica elettronica è piombato nella più grande confusione poiché a nessuno, all'inizio, era ben chiaro cosa poter fare con questo tipo di interfaccia. In parte è dovuto anche al fatto che gli utenti (musicisti e non tecnici elettronici) sono principalmente interessati ai risultati musicali (e giustamente!) e non ai particolari tecnici per raggiungere tali risultati.

Durante la descrizione del progetto qui presentato, vengono fornite interessanti informazioni tecniche (quelle di cui dispongono i progettisti degli apparecchi elettronici dotati di MIDI) e sono facilmente comprensibili a chi abbia una certa esperienza di programmazione.

Normalmente l'interfaccia comprende due sezioni: una per la ricezione ed una per la trasmissione dei dati. Ognuna di queste ha un proprio connettore, rispettivamente etichettato MIDI IN e MIDI OUT. Opzionalmente può trovarsi un terzo connettore, etichettato MIDI THRU, che si limita ad emettere un'esatta copia dei segnali ricevuti da MIDI IN. Questo risulta utile per facilitare il collegamento "in serie" di più apparecchi senza dover aggiungere ulteriori prese MIDI OUT sul trasmettitore.

Prima di tutto occorre individuare la natura dei dati scambiati tramite la MIDI, e questo è reso più semplice se si considerano i vari strumenti elettronici, nel sistema musicale, collegati come particolari periferiche di un computer. La distinzione tra alcuni di questi dispositivi può non essere immediatamente evidente: è il caso dei più moderni sintetizzatori che, sebbene formati da due diversi elementi (la tastiera per l'immissione ed il

generatore di suoni per l'emissione), formano un solo strumento. Il collegamento più ovvio, in tal caso, è quello diretto: premendo un tatso della tastiera si ottiene un suono dal generatore.
Questo sistema è paragonabile ad una sofisticata macchina da scrivere elettronica in cui due componenti distinte, la tastiera ed il dispositivo di stampa, possono essere collegate in modo da riprodurre immediatamente quanto digitato. Tuttavia il collegamento può avvenire secondo un criterio ben diverso e ben noto a chi usa abitualmente un word processor: la tastiera è collegabile ad un computer che elabora lo scritto prima di inviarlo alla stampante.

Nel caso di un sintetizzatore è importante notare che l'interfaccia interna, tra la tastiera ed il generatore di suoni, opera esclusivamente su segnali digitali. In sostanza, la tastiera agisce la puro e semplice controllo esattamente come in un word processor la tastiera controlla il dispositivo di stampa.
Anche in un sintetizzatore musicale, dunque, è possibile inserire un computer tra la tastiera ed il generatore di suoni. La musica, in questo caso, non viene rappresentata da variazioni di pressione nell'aria (come negli strumenti tradizionali), bensì da una sequenza di "eventi" codificati tramite la tastiera e, in seguito, inviati al sistema di riproduzione. La conversione di tali segnali sonori, in effetti, spetta ai circuiti del generatore di suoni per poi, successivamente, trasmessi ad un amplificatore audio. Di cinseguenza la MIDI introduce, per la prima volta nella storia della musica, la prospettiva di utilizzare sistemi di controllo svincolati dal concetto di "tastiera musicale".
Un buon esempio (se pur rudimentale) di un conosciutissimo sistema di elaborazione musicale, risale a moltissimi anni fa: la pianola. Questo strumento era uno speciale tipo di pianoforte in grado di suonare brani musicali codificati sotto forma di fori su di un rullo di carta. Un meccanismo di lettura, attraverso il quale passava a velocità costante il rullo codificato, provvedeva a convertire il codice, in combinazioni di note agendo sui rispettivi martelletti del pianoforte. La distanza di ciascun foro, dal margine laterale del foglio di carta, corrispondeva all'altezza della nota, mentre la distanza, in senso longitudinale, corrispondeva all'intervallo di tempo tra una nota e l'altra. Ciò consentiva una forma piuttosto elementare di registrazione musicale ma consentiva apportare tutte le correzioni ed aggiunte desiderabili: basta modificare la posizione di

uno o più fori nel rotolo di carta. Questo sistema ha notevoli analogie con le schede perforate usate nella programmazione dei primi computer.
Un attuale sistema di elaborazione musicale (il termine “music processor” è ormai di uso corrente) può essere considerato, quindi, una versione rivista ed aggiornata del vecchio metodo per pianola: la memoria del computer (o quella dei dischi) ha sostituito la carta mentre, un sintetizzatore controllato da un’interfaccia, ha rimpiazzato il pianoforte. La revisione di un pezzo musicale, oggi, viene fatta direttamente nella memoria del computer grazie anche all’apposito software. Prima di giungere a tutto questo, comunque, occorre essere a conoscenza di due cose riguardanti la MIDI: il tipo di trasmissione dai dati impiegato ed il formato di tali dati. Questi due elementi corrispondono alle specifiche hardware e software dell’interfaccia. Per adesso consideriamo l’aspetto hardware.

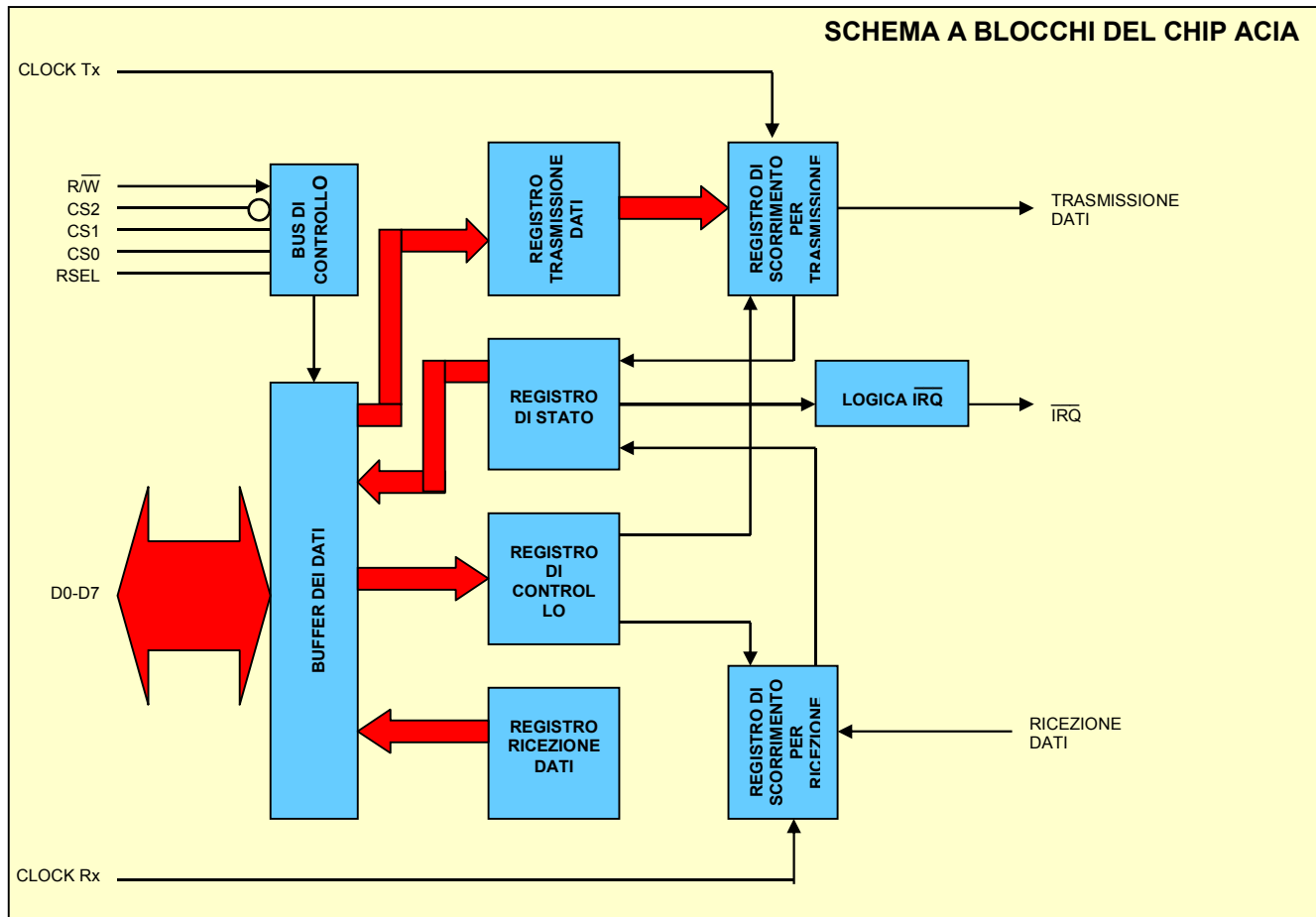
Trasmissione seriale dei dati

Innanzitutto è necessario sapere come viene inviato un byte da una parte all’altra del sistema. La MIDI funziona come un’interfaccia seriale asincrona, con frequenza di lavoro a 31.25 Kbaud. Ciò significa che può essere inviato un solo bit alla volta e che il dispositivo trasmettitore e quello ricevente non sono sincronizzati tra loro mediante un comune segnale di temporizzazione. Lo svantaggio di una simile scleta è la relativa lentezza ma, in compenso, bastano due sole linee di comunicazione che rendono il tutto molto economico e facile da realizzare.
Le specifiche MIDI stabiliscono l’impiego di normali connettori DIN a 5 poli (disposti a 180°) o, in alternativa, di connettori professionali XLR a 3 poli. Questa scelta è stata fatta intenzionalmente perché, il costo irrisorio di questi componenti, ne favorisca la diffusione ad un maggior numero possibile di strumenti.
La critica più frequente rivolta alla MIDI riguarda la sua lentezza che, in sistemi di notevoli dimensioni (come si vedrà in seguito), può anche influire sensibilmente. D’altra parte è proprio grazie a questa relativa lentezza che è possibile utilizzare, per il controllo di un sistema MIDI, anche comuni home computer.
I dati vengono inviati in gruppi di 10: un byte di dati effettivi, più un bit di start ed uno di stop. Normalmente, in assenza di segnale, la linea si trova a livello logico alto (+5V per la MIDI). L’inizio della trasmissione viene segnalato da un passaggio al livello basso (0V) per la durata di un bit (bit di start). In seguito vengono trasmessi in sequenza gli otto bit di dati (partendo dal bit più significativo), seguiti da un passaggio a livello alto per la durata di un bit (bit di stop). L’invio di un dato, pertanto, richiede una durata di 10 bit, ossia 10/31.25 Kbaud = 320 microsecondi (μs). Questo valore è di capitale importanza per progettare un sistema MIDI come si avrà modo di constatare in seguito.
In assenza di segnale, come già detto, la linea è costantemente a livello alto ed il ricevitore verifica, a intervalli frequentissimi, se c’è stato un cambiamento di stato: un passaggio a livello basso indica l’inizio di una trasmissione. A questo punto, il ricevitore sa di dover contare 8 bit di dati e predispone un contatore che attende per un periodo pari ad un bit e mezzo. Al termine di questo, il ricevitore è in grado di stabilire se il bit ricevuto è a livello alto oppure basso. I successivi bit vengono letti ad intervalli regolari.
Alla fine viene verificata la presenza del bit di stop (livello alto) che serve ad indicare la completezza del dato ricevuto. Ricorrendo a questo criterio di trasmissione, non occorre che i segnali di temporizzazione siano perfettamente sincronizzati; il che semplifica l’impiego dell’interfaccia. Ovviamente le differenze non possono essere di ordine troppo grande: è stata stabilita una tolleranza dell1%.
L’interfaccia, considerando la sua natura seriale, è relativamente veloce (la RS-232, ad esempio, ha una velocità massima di 19.2 Kbaud), quindi i cavi di collegamento non dovrebbero superare i 15 metri per evitare le disperioni del segnale.

Ritardi di trasmissione

A questo punto conviene esaminare la principale limitazione della MIDI, ossia il ritardo di trasmissione tra i vari strumenti. In generale, un messaggio di nota attiva/non attiva consiste in tre byte il cui tempo necessario per l’invio è di 3x320 μs = 0.96 millisecondi (ms). Adesso si consideri un sistema con un sequenziatore multicanale in cui, ad esempio, occorra inviare 20 messaggi simultanei: l’ultima sequenza viene inviata con un ritardo di circa 20 ms rispetto alla prima!
In molti sistemi sofisticati, i segnali audio prodotti hanno tempi d’innesco sull’ordine dei 2-3 ms per cui, anche un orecchio non particolarmente addestrato, può accorgersi di un ritardo di ben 20 ms.

Il progetto hardware dell'interfaccia MIDI per strumenti musicali

Il componente principale dell'interfaccia è un circuito ACIA (Asynchronous Communications Interface Adaptor) che converte i dati paralleli del computer nel formato seriale, richiesto dalla MIDI, e viceversa. Di solito l'ACIA è un unico circuito integrato che, nel nostro caso, è il chip MC6850 utilizzato su molti home computer per gestire lo scambio asincrono di dati tra cassette e le interfacce del tipo RS-232. Altri componenti del progetto sono: un clock d'interfaccia, un accoppiatore ottico e tre prese d'ingresso e di uscita. Le necessarie connessioni alla porta del computer, sia per il Commodore 64 che per il Micro BBC, vengono descritte in seguito.

Internamente il circuito ACIA contiene una serie di registri a 8 bit. La linea d'ingresso per la selezione del registro (RSEL) permette di accedere ai registri del bus dei dati. Per utilizzare il software di controllo, fornito in seguito, non è indispensabile comprendere il funzionamento dei registri dell'ACIA. Tuttavia, per coloro che volessero cimentarsi per conto proprio nella programmazione dell'interfaccia, ecco le informazioni necessarie:

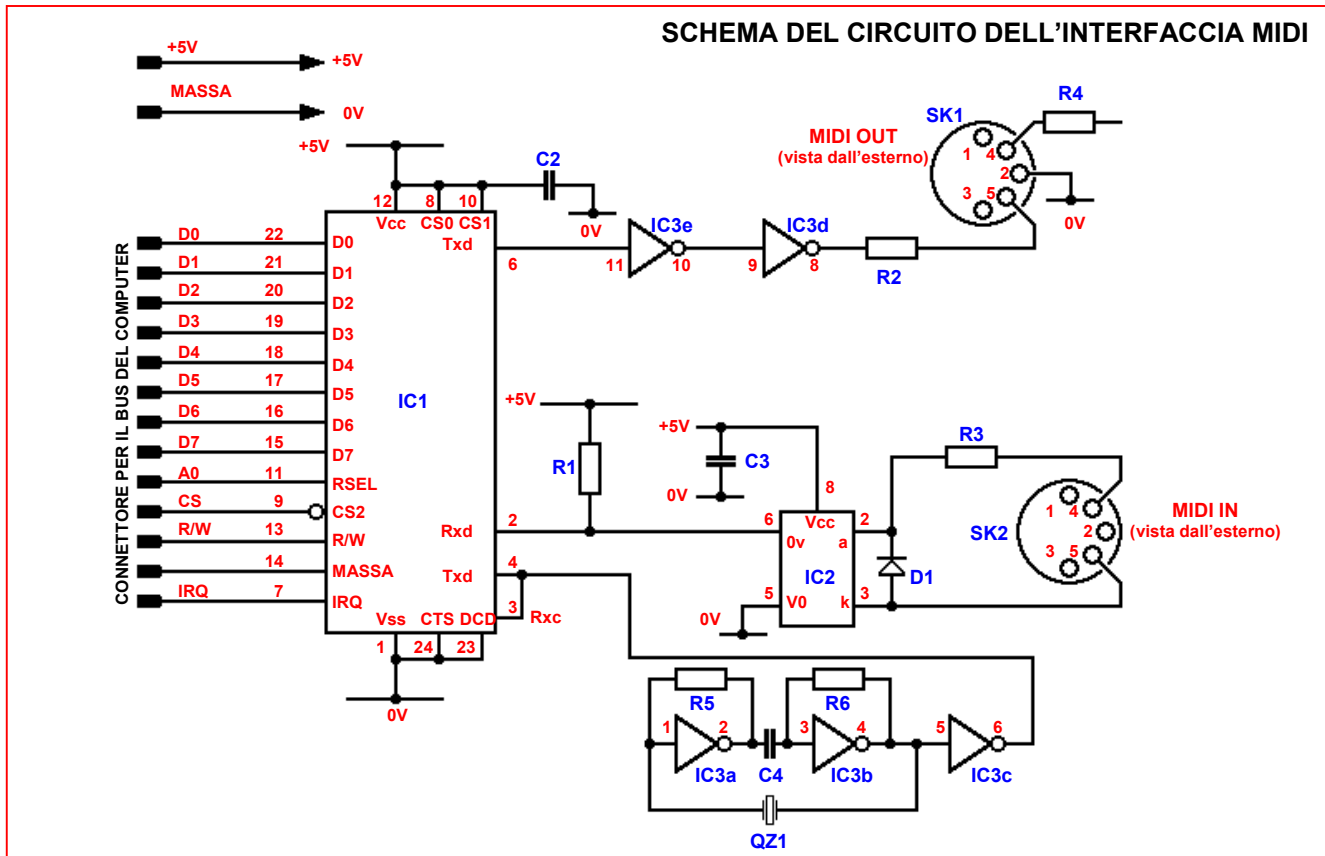
- Il "registro di stato" è composto da 8 bit che descrivono lo stato attuale del circuito ACIA. Si può esaminarne il contenuto con una "lettura" dell'ACIA dopo aver posto la linea RSEL a livello logico 0.
- Il "registro di controllo" contiene 8 bit che controllano il funzionamento dell'ACIA. Per accedere al registro occorre eseguire un'operazione di "scrittura" sull'ACIA con la linea RSEL a livello logico 0.
- Il "registro di scorrimento per la trasmissione" (TSR) esegue la conversione da parallelo a seriale, necessaria per trasmettere un byte di dati. Detto registro riceve un byte dal "registro di trasmissione dei dati" (TDR) ogni volta che, quest'ultimo, è completato: purché sia conclusa la trasmissione del byte precedente. Ciò pone a 1 il bit 1 del registro di stato. Il byte, in seguito, viene scaricato dal registro alla velocità indicata dal clock di trasmissione ed è, in questa fase, che ai suoi 8 bit vengono aggiunti i bit di start e di stop. Il TSR è un registro "interno" e non può essere raggiunto direttamente attraverso il bus dei dati.
- Il "registro di trasmissione dei dati" (TDR) è un 'buffer' (deposito temporaneo di memoria) interposto tra il TSR ed il bus dei dati del sistema. Il byte da trasmette viene caricato in questo registro, eseguando una "scrittura" sull'ACIA, dopo aver posto al valore logico 1 la linea di selezione del registro (RSEL). Tale operazione azzera il bit 1 del registro di stato. Quindi, per non perdere informazioni, il TDR non deve mai

essere caricato se il suo bit di stato è a 0 poiché ciò indica che il byte precedente si trova ancora all'interno del TDR, in attesa di venire trasmesso.

- Il "registro di scorrimento per la ricezione" (RSR) esegue la conversione da seriale a parallelo per la ricezione dei dati dal sistema MIDI. Dopo aver ricevuto un byte completo, il registro lo trasferisce all'RDR e pone ad 1 il bit di stato 0. Se il bit conteneva già il valore, l'RSR seleziona anche il bit 5 per indicare che il byte precedente non era stato letto dalla CPU e che l'informazione è perduta.

  Se il byte ricevuto manca dei bit di start e di stop, il registro seleziona il bit di stato 4; ciò può verificarsi quando la linea d'ingresso seriale è disturbata da un 'rumore elettrico'.

  Come il TSR, anche l'RSR non può essere raggiunto direttamente dal bus.
- Il "registro di ricezione dei dati" viene caricato ogni volta che l'RSR riceve un byte completo: quindi contiene l'ultimo byte ricevuto. Anche in questo caso, per accedere al registro, occorre eseguire una "lettura" del chip ACIA, ma con la linea RSEL al valore logico 1.

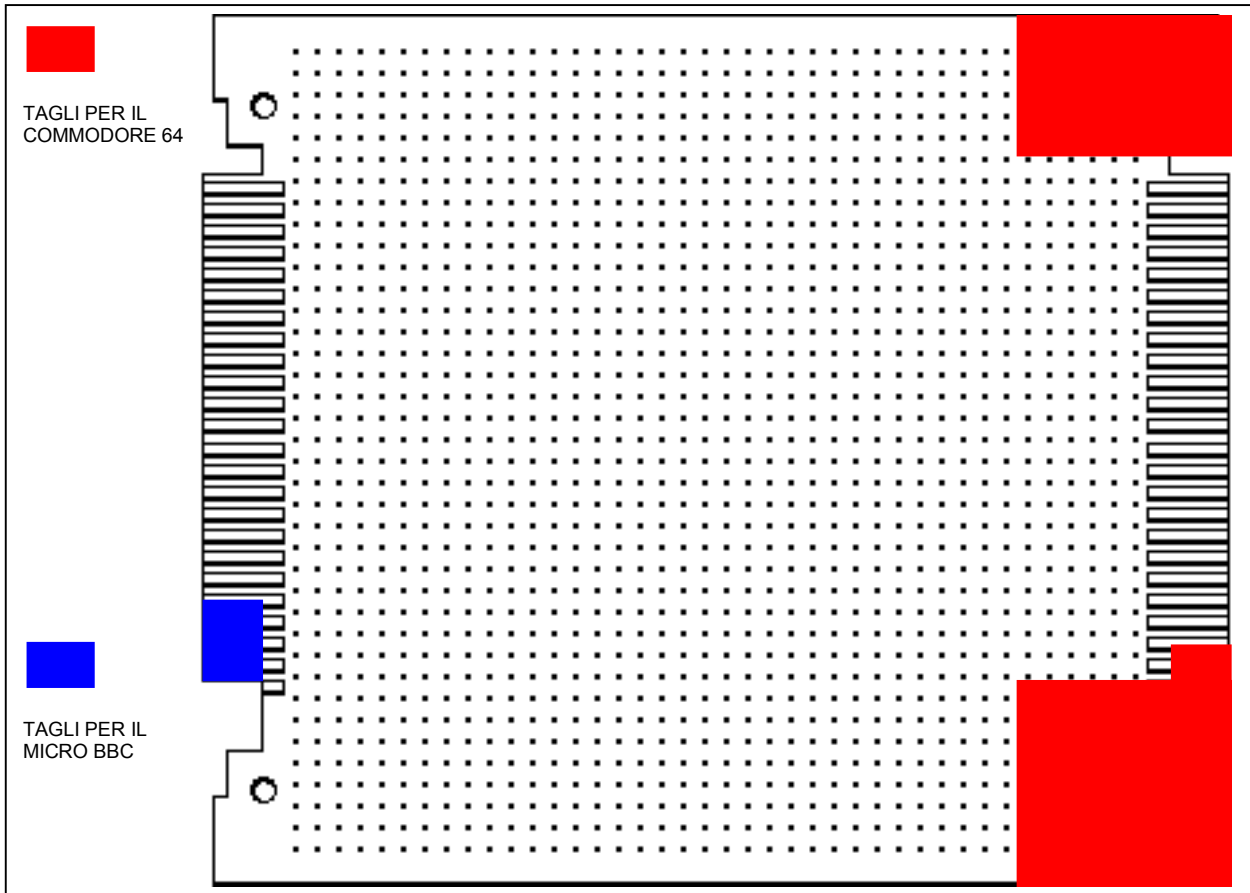
  L'operazione azzera il bit di stato 0.

Collegamenti del circuito

L'accoppiamento ottico è ottenuto con l'optoisolatore 6N139 (IC2). I segnali del clock, per la trasmissione, non vengono generati dal clock del computer ma da un oscillatore a 2 MHz realizzato con il quarzo e con IC3a e IC3b. IC3c, IC3d ed IC3e forniscono le funzioni di bufferizzazione per il trasmettitore ed il generatore di clock. Poiché il chip ACIA MC6850 è compatibile con il bus della CPU 6502, è possibile collegare i micro, basati sul 6502, alle linee di Lettura/Scrittura, dell'IRQ e del Clock. Il canale d'ingresso del clock per l'ACIA è stato etichettato con la sigla E per evitare confusioni con i canali di trasmissione del clock TXD e TXC. Dato che le linee di selezione del chip, CS0 e CS1, sono collegate alla linea di alimentazione dei 5 volt, risultano sempre a livello logico alto. Il solo canale utile per la selezione del chip è CS2. Tale linea viene posta a livello basso da una serie di indirizzi per consentire sia l'esecuzione di operazioni di lettura e di scrittura sul bus dei dati, sia l'accesso ai registri interni dell'ACIA.

ELENCO DEI COMPONENTI

R1	270 ohm	QZ1	cristallo da 2.0 MHz
R2	220 ohm	IC1	chip ACIA MC68B50
R3	220 ohm	IC2	chip optoisolatore 6N139
R4	220 ohm	IC3	invertitore esadecimale 74LS04
R5	680 ohm	SK1	presa DIN a 5 piedini (a 180°)
R6	680 ohm	SK2	presa DIN a 5 piedini (a 180°)
C2	100 nF policarbonato		
C3	100 nF policarbonato	1	zoccolo DIL a 24 piedini
C4	1 nF policarbonato	1	zoccolo DIL a 8 piedini
D1	1N914	1	zoccolo DIL a 14 piedini
		1	scheda con due connettori a pettine per l'assemblaggio dei componenti

TAGLI PER IL COMMODORE 64

TAGLI PER IL MICRO BBC

CABLAGGIO DEL CIRCUITO

PRESE DIN A 5 POLI (A 180°)

MIDI IN

MIDI OUT

220 ohm

220 ohm

2 MHz

1nF

1N914

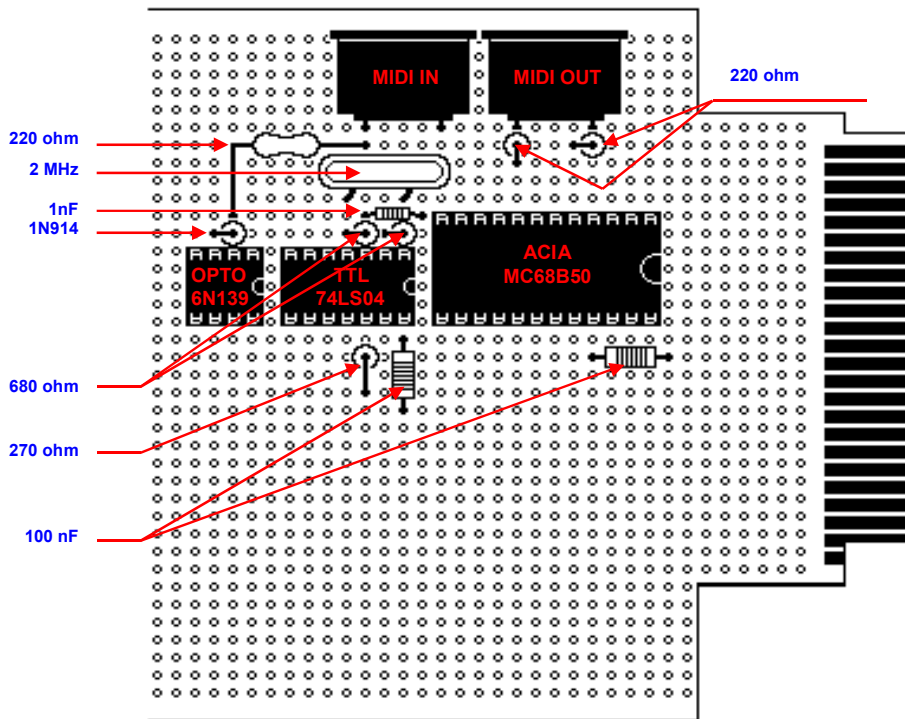
OPTO 6N139

TTL 74LS04

ACIA MC68B50

680 ohm

270 ohm

100 nF

Realizzazione della MIDI
I componenti principali dell'interfaccia MIDI vengono montati su di una speciale scheda prestampata dotata, alle due estremità, di una coppia di connettori a pettine ramati su entrambe le facce: quello di destra collega la MIDI al Commodore 64, quello di sinistra al Micro BBC. Prima di montare i componenti, occorre tagliare e rimuovere alcune parti della scheda. I tagli vanno eseguiti solo sull'estremità della scheda relativa al computer che si vuol collegare all'interfaccia.
I ponticelli, realizzati usando un normale filo elettrico per "Wire wrap" (ossia molto sottile), vanno montati sul lato componenti della scheda (il lato che si vede nel disegno sopra), saldando le due estremità sulle piste ramate sottostanti.

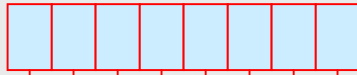
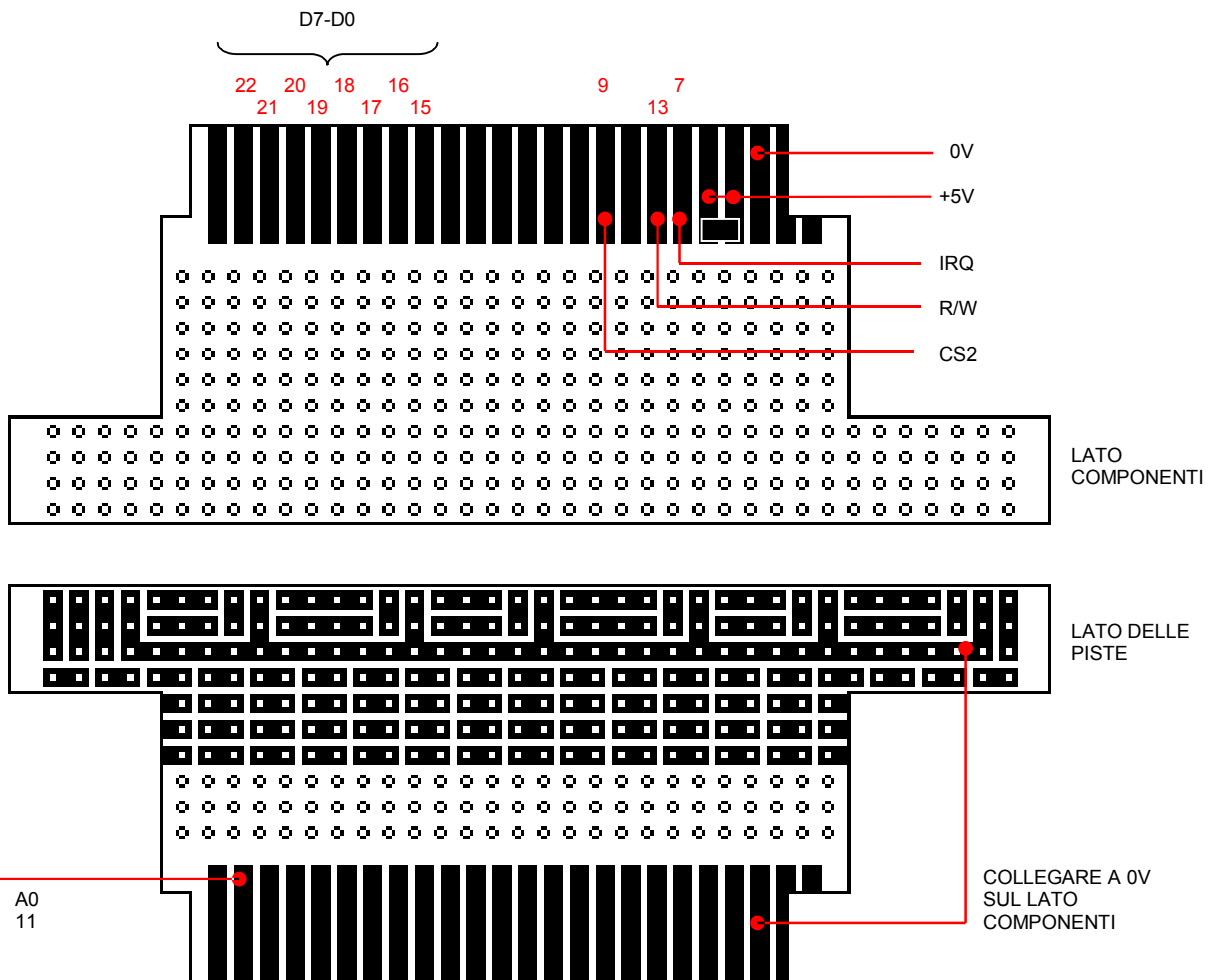
Linee di melodia

Prosegue il progetto dell'interfaccia MIDI: il collegamento del chip ACIA

Il "cuore" dell'interfaccia MIDI è il chip ACIA compatibile con i processori 6502 e 6510 usati, rispettivamente, nel Micro BBC e nel Commodore 64. È quindi possibile collegare le linee di controllo, d'indirizzamento e dei dati del processore, direttamente ai piedini di questo chip.
La scheda per il Commodore 64 è progettata per l'innesto alla porta d'espansione del computer, mentre quella per il Micro BBC viene collegata alla porta "tube", situata sul retro della macchina, tramite un cavo a nastro da 40 canali dotato di un apposito connettore.
A questo proposito mi sento in dovere di fare una promessa: tutti i dati che sto raccogliendo in questo mio file e che si intitola "Il Commodore 64", riguardano, appunto, questo home computer perché ne sono in possesso. Non me ne vogliano i possessori del Micro BBC ma, da qui in avanti, tratterò solo le informazioni che mi riguardano: quelle per il Commodore 64.
Realizzata la scheda, occorre verificarne il funzionamento eseguendo alcune prove quali il caricamento di dati nel chip ACIA e l'esecuzione di un semplice ciclo di controllo.
Per accedere alle periferiche collegate al bus dei dati principali, ma che non prevedono lo stesso numero di linee d'indirizzo rispetto alla CPU, è necessaria una decodifica degli indirizzi. Di solito tali periferiche possiedono un piedino di "selezione del chip" che, se selezionato correttamente, consente di usare il bus dei dati; di conseguenza le linee di indirizzamento, collegate alla periferica, permettono la selezione dei diversi registri.
I segnali di selezione del chip, per gli svariati apparecchi collegati al sistema, vengono generati decodificando le linee più significative degli indirizzi: le combinazioni di bit inviate su di esse possono essere quindi impiegate per selezionare una particolare periferica. In questo modo i registri interni dell'apparecchio entrano a far parte della mappa di memoria della CPU ed è possibile accedervi e manipolarli come fossero normali locazioni di memoria.
Naturalmente ci si deve accertare che gli indirizzi, usati dalla periferica, non corrispondano ai registri utilizzati dal sistema operativo del computer.
La porta d'espansione del Commodore 64 possiede 2 uscite, indicate dalle sigle I/01 e I/02, che vengono poste a livello logico "basso" quando il sistema accede alle pagine $DE e $DF. Collegando la linea CS2 del chip ACIA all'uscita I/01, è possibile "impaginare" il chip nella pagina di memoria $DE. Poiché l'ACIA non è collegato alle linee d'indirizzo A1-A7, i suoi registri interni possono essere raggiunti usando un qualsiasi indirizzo compreso tra $DE00 e $DEFF. Collegando direttamente A0 al piedino di selezione del registro dell'ACIA, tutti gli indirizzi di valore pari permettono di accedere ai registri di trasmissione/ricezione dei dati. La scelta più ovvia, comunque, è quella degli indirizzi $DE00 (56832 decimale) e $DE01 (56833 decimale).

Programmazione dell'ACIA

Per programmare l'interfaccia MIDI si deve conoscere il funzionamento dei quattro registri del chip ACIA. Quanto segue si riallaccia alla descrizione già fatta nel capitolo precedente e si riferisce alle funzioni dei bit dei registri di controllo e di stato. Intervenendo sul registro di controllo, si può programmare il chip ACIA, in modo che generi un'interruzione per la CPU, qualora vengano selezionati particolari bit di stato all'interno dei circuiti di ricezione e/o trasmissione dell'ACIA. Quando il bit 5 e 6 del registro di controllo vengono posti, rispettivamente, ad 1 ed a 0 ed il bit di stato TDR si trova a livello basso, il chip ACIA genera una interrupt di trasmissione. Se invece vengono posti a 1 il bit 7 del registro di controllo 7 ed il bit 0 del registro di stato, l'ACIA genera una interrupt di ricezione. Quest'ultima viene annullata con una lettura del registro di ricezione dei dati (RDR) purché non si trovi a livello alto il bit 5 del registro di stato che viene azzerato leggendo, prima dell'RDR, lo stesso registro di stato. Anche il bit 2 del registro di stato può generare una interrupt, ma non viene utilizzata in questo progetto.
Il chip prevede due linee separate per la trasmissione e per la ricezione dei segnali di sincronizzazione (clock) ma, in genere, vengono collegate fra loro. Il clock che regola la frequenza di baud viene generato dal segnale di clock in ingresso (che può essere anche diviso per 16 o 32 a seconda del valore dei bit di controllo 0 e 1).
I bit 2, 3 e 4 del registro di controllo fissano, rispettivamente: il numero di bit di stop, il numero di bit nei dati e stabiliscono se la parità è pari, dispari oppure se non prevista. Per adattare il circuito alla standard MIDI, si deve porre a 1 i bit 2 e 4 ed azzerare il bit 3 (8 bit di dati, nessuna parità, 1 bit di stop).
I bit 5 e 6 del registro di controllo regolano il funzionamento della trasmissione. Per quanto riguarda il sistema MIDI, l'azzeramento del bit 6 consente al bit 5 di abilitare le interrupt del trasmettitore.

I REGISTRI DELL'ACIA
REGISTRI DI TRASMISSIONE/RICEZIONE DEI DATI:
SELEZIONE DEL REGISTRO = 1
RICEZIONE – LETTURA DEL REGISTRO
TRASMISSIONE – SCRITTURA DEL REGISTRO
REGISTRO DI STATO (SOLO LETTURA)
SELEZIONE DEL REGISTRO = 0
REGISTRO DI RICEZIONE DEI DATI PIENO
REGISTRO DI TRASMISSIONE DEI DATI VUOTO
INDIVIDUA LA PORTANTE DEI DATI
PRONTO PER LA TRASMISSIONE
ERRORE DI FORMATO
SOVRACCARICO DEL RICEVITORE
ERRORE DI PARITÀ (*)
RICHIESTA D'INTERRUZIONE
REGISTRO DI CONTROLLO (SOLO SCRITTURA)
SELEZIONE DEL REGISTRO = 0
SELEZIONE DEL CONTATORE DI DIVISIONE
00 DIVIDE PER 1 (*)
01 DIVIDE PER 16 (*)
10 DIVIDE PER 64
11 AZZERAMENTO PRINCIPALE
SELEZIONE DELLA PAROLA
BIT DEI DATI BIT DI STOP PARITÀ
000 7 2 PARI (*)
001 7 2 DISPARI (*)
010 7 1 PARI (*)
011 7 1 DISPARI (*)
100 8 2 NESSUNA (*)
101 8 1 NESSUNA
110 8 1 PARI (*)
111 8 1 DISPARI (*)
CONTROLLO DELLE INTERRUPT DI TRASMISSIONE
00 DISABILITA (*)
01 ABILITA
10 DISABILITA (*)
11 DISABILITA (*)
CONTROLLO DELLE INTERRUPT DI RICEZIONE
0 DISABILITA
1 ABILITA
(*) = NON APPLICABILE AL PROGETTO MIDI

COLLEGAMENTI A PETTINE

Lo schema qui sopra illustra i collegamenti dei piedini del chip ACIA per il Commodore 64. I numeri assegnati ai cavetti saldati ai connettori corrispondono ai piedini dell'ACIA. Terminato il montaggio, innestare la scheda nella porta d'espansione del Commodore 64 con il lato dei componenti rivolti verso l'alto ed accendere il computer per collaudarne il funzionamento.

Collaudo della scheda

Innestata la scheda, conviene eseguire alcune prove per controllarne il funzionamento. Se una di queste dovesse fallire, si può utilizzare un multimetro per individuare gli eventuali difetti. Se lo strumento non dovesse rivelare alcuna anomalia, sarà necessario effettuare un'accurata ispezione visiva della scheda.

1) Se il computer non funziona normalmente con la scheda inserita:
   verificare che la tensione tra le linee +5V e 0V sia effettivamente di 5 volt. In caso negativo controllare che tutti gli integrati siano inseriti correttamente e che non vi siano cortocircuiti tra le piste d'alimentazione della scheda.
   disinserire la scheda e, con un multimetro, controllare che non vi siano cortocircuiti tra i collegamenti sul bus di comunicazione con il computer.

2) Se il computer funziona normalmente, collegare le prese MIDI IN e MIDI OUT dell'interfaccia con dei comuni cavetti hi-fi a 5 piedini e digitare il seguente comando:

```
POKE 56832,3
```

che carica il valore 3 nel registro di controllo dell'ACIA e produce un reset generale del sistema. L'istruzione:

POKE 56832,22

carica il valore $16 nel registro di controllo e configura l'ACIA nel seguente modo:

- disabilita le interrupt di trasmissione e di ricezione (dato che il sistema non è ancora in grado di gestirle)
- assegna, alle parole seriali trasmesse e ricevute, una lunghezza di 8 bit (per i dati) più un bit di stop senza alcuna generazione/verifica della parità
- definisce, come frequenza di baud, l'impulso di sincronizzazione (clock) ai piedini 3 e 4 diviso per 64 (2MHz/64 = 31,25 KHz che è la frequenza stabilita dallo standard del sistema MIDI).

A questo punto l'ACIA dovrebbe essere in grado di ricevere e trasmettere dati.
Si verifichi il funzionamento leggendo il registro di stato con l'istruzione:

PRINT PEEK(56832)

Il valore letto dovrebbe essere 2. Questo indica che sia il registro di trasmissione dei dati (bit 1 posto a 1) sia il registro di ricezione dei dati (bit 0 posto a 0) sono vuoti. Poiché il computer non ha ricevuto alcun dato e le interrupt sono disabilitate, i bit di stato, compresi tra 2 e 7, dovrebbero essere a zero.

3) Trasmettere un byte dal registro di trasmissione a quello di ricezione con il comando:

POKE 56833,X

dove X è un numero compreso tra 0 e 255. Questa istruzione carica nel registro dei dati un valore da trasmettere.

4) Il byte viene ricevuto in una frazione del tempo necessario per inserire il cussessivo comando per la lettura del registro di stato:

PRINT PEEK(56832)

Il nuovo valore dovrebbe essere 3. Il bit 1, azzerato subito dopo l'ultimo comando, viene posto nuovamente a 1 non appena il registro di trasmissione dei dati risulta disponibile per il byte successivo. Il bit 0 viene posto a 1 per indicare che il bit è stato ricevuto e può essere letto dal registro dei dati. In questo caso può verificarsi un errore: se il bit 0 non risulta ad 1, probabilmente il circuito del canale di trasmissione non è elettricamente chiuso e, quindi, l'ingresso del ricevitore rimane a livello alto ampedendo la lettura.

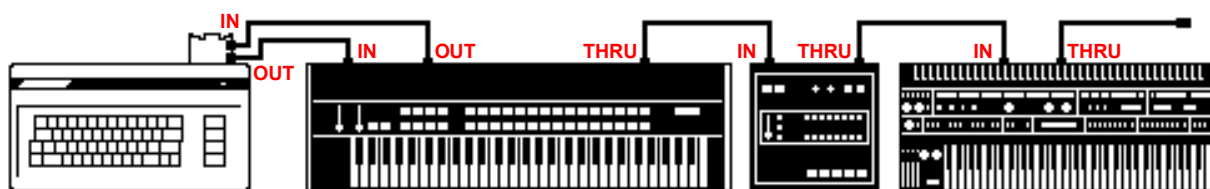
Accertata la ricezione di un byte, il seguente comando, per la lettura del registro dei dati, verifica se il byte ricevuto corrisponde a quello trasmesso:

PRINT PEEK(56833)

Questa operazione di lettura dovrebbe fornire lo stesso valore X trasmesso in precedenza. Conviene ripetere le fasi 3, 4 e 5 usando valori diversi per X.

Regole di composizione

Esaminiamo il software necessario per l'interfaccia MIDI

Come inviare un byte di dati ad uno strumento, attraverso la MIDI, è già stato spiegato; adesso occorre determinare il formato del byte (o dei byte) occorrenti per comunicare le informazioni mecessarie. Un singolo "evento" musicale viene trasmesso sotto forma di "messaggio", ossia come un gruppo di byte. La maggior parte di questi messaggi è lunga due o tre byte, con l'eccezione dei messaggi "esclusivi di sistema", esaminati più avanti, che possono contenere un qualsiasi numero di byte. Ogni messaggio inizia con un byte il cui bit più significativo (BPS) è posto a 1. Seguono i restanti byte del messaggio che hanno il BPS posto a 0. I byte che possiedono il BPS pari a 1, vengono definiti "byte di stato", gli altri semplicemente "byte di dati". I messaggi MIDI sono suddivisi in due categorie principali: quelli "di canale" e quelli "di sistema". La necessità di avere messaggi di canale deriva dall'utilizzo di collegamti "in cascata", adottati principalmente nei sistemi più piccoli. In tal caso, ogni unità riceve tutti i segnali emessi dall'unità trasmittente, ma riconosce ed esegue solo quelli relativi al "numero di canale" assegnatole (da 1 a 16).

Passa parola...
Il metodo più diffuso, di collegare le varie apparecchiature compatibili con il sistema MIDI, è quello chiamato "in cascata" che si avvale, in particolare, della presa identificata con MIDI THRU. Questa consente una connessione in cui le linee di trasmissione sono comuni a tutti gli strumenti; questi ultimi vengono selezionati in base a specifici codici di identificazione.

I messaggi di canale sono composti da un byte di stato il cui valore è compreso tra $80 e $EF, e da uno o due byte di dati. Nel byte di stato il numero di canale viene codificato nei quattro bit meno significativi (BMS) del byte, pertanto, il valore $00 corrisponde al canale 1 e $0F corrisponde al canale 16. I rimanenti tre bit determinano il tipo di messaggio che segue. Una caratteristica speciale dei messaggi di canale è che non occorre ripetere il byte di stato nel caso in cui questo fosse identico a quello trasmesso nel messaggio precedente. Pertanto, una volta inviato un byte di stato, questo rimane attivo finché non ne viene indicato un altro (l'unica eccezione si presenta quando un messaggio di sistema, in tempo reale, interrompe temporaneamente lo stato "attuale").
Questo criterio di funzionamento è particolarmente comodo se si tratta di inviare, consecutivamente, più messaggi per l'emissione (on) o l'annullamento (off) di note. L'annullamento di una nota, in particolare, si ottiene specificando una "velocità" pari a zero. La possibilità di mantenere inalterato il byte di stato consente, dunque, di risparmiare memoria e di velocizzare l'emissione delle note.
L'assegnazione della "voce", nei sintetizzatori, è il processo che consente di dirigere un messaggio (sia proveniente dal sistema MIDI che da tastiera) ad uno dei sintetizzatori collegati al sistema. Se, per esempio, un sintetizzatore polifonico permette la simultanea emissione di sei note, si dice che ha sei voci. Per controllare la risposta dello strumento, al messaggio inviatogli, sono disponibili svariati messaggi modali MIDI. Un messaggio modale, comunque, dovrebbe essere ignorato da uno strumento ricevitore che non sia capace di operare nel modo specificato.

I messaggi modali MIDI

Il modo più semplice è l'omni mode in cui il ricevitore risponde a tutti i messaggi di canale, qualunque sia il numero di canale specificato nel messaggio. Se l'omni mode viene disattivato, l'unità risponde solo ai messaggi per il canale MIDI al quale è associato.
La maggior parte dei sintetizzatori polifonici ha un numero limitato di voci (di solito sei oppure otto). Ciò significa che occorre prevedere il caso in cui venga richiesta l'emissione di una nota quando tutte le voci sono già state impegnate da altri messaggi.

Di solito l'assegnamento delle note avviene con un criterio di rotazione in modo che, la nota "più vecchia", venga annullata per procedere all'emissione di quella più recente. Questo criterio viene selezionato abilitando il modo poly.

L'abilitazione del modo mono, invece, predispone il dispositivo ricevente in maniera da assegnare, ciascuna delle sue voci, ad uno tra un gruppo di canali MIDI consecutivi, iniziando dal canale principale (o fondamentale) del sintetizzatore). Il messaggio del modo mono viene inviato sul canale principale ed il secondo byte, di tale messaggio, specifica quanti canali sono richiesti.

Se è attivo il modo omni, un messaggio mono istruisce semplicemente il ricevitore ad assegnare una voce monofonica ai messaggi di canale MIDI, su tutti i canali.

Una "star" d'eccezione
Le prime implimentazioni del sistema MIDI non prevedevano la selezione tramite numero di canale. Un collegamento "in cascata" non aveva quindi alcun senso, poiché tutti gli strumenti tendevano a rispondere ai messaggi emessi dal trasmettitore. L'unica forma di collegamento utile era quindi quella detta "a stella" al centro della quale si trova l'unità centrale di trasmissione, dotata di un'uscita per ciascuna unità periferica.

I *messaggi di sistema* non contengono specifiche di canale nel byte di stato, così il loro invio riguarda tutti gli strumenti collegati al sistema. I messaggi si dividono in tre categorie: i messaggi comuni, in tempo reale ed esclusivi.

I *messaggi di sistema comuni* sono composti da un byte di stato (che può assumere valori compresi tra $F1 e $F7) e da 0, 1 o 2 byte di dati.

I *messaggi di sistema in tempo reale* riguardano tutti i componenti del sistema e sono composti soltanto da un byte di stato (di valore compreso tra $F8 e $FF) e nessun byte di dati. Questa particolarità ne permette l'invio in qualsiasi momento, anche interrompendo altri messaggi in corso d'esecuzione.

Principalmente questi messaggi servono per la sincronizzazione di dispositivi quali sintetizzatori di batteria e sequenziatori. La maggior parte dei normali sintetizzatori, comunque, ignora i messaggi di questo tipo se non possiedono, al loro interno, un sequenziatore capace di sincronizzarsi con il sistema MIDI.

I *messaggi di sistema esclusivi* iniziano con un byte di stato il cui valore è $F0, seguito da un qualsiasi numero di byte di dati. Il messaggio può terminare con il byte di fine-messaggio $F7 o con un qualsiasi

altro byte di stato. Il primo byte di dati è il codice identificativo (ID) del fabbricante: se esso non corrisponde al codice dell'apparecchio ricevente, quest'ultimo deve ignorare il messaggio.

I messaggi di questo tipo servono per scmbiare informazioni tra strumenti dello stesso genere. I dati trasmessi, in genere, costituiscono particolari sequenze di programmazione ("patch") per sintetizzatori. Questa trasmissione non deve essere confusa con il messaggio di canale $Cx che opera una selezione tra i programmi patch già residenti nella memoria del sintetizzatore ricevente.

Il contenuto dei messaggi di sistema esclusivi, di solito, compete ai fabbricanti purché sia stato assegnato loro un codice valido di identificazione.

MESSAGGI PER I CANALI DELLE VOCI

Stato	Dato1	Dato2	Descrizione	Note
$8x	p-	v	Nota off p = altezza v = velocità di off	1,2
$9x	p	v	a) v > 0 Nota on p = altezza v = velocità	1,2
			b) v = 0 Nota off p = altezza	1,2
$Ax	p	pr	Post-pressione polifonica p = altezza pr = pressione	1,3a
$Bx	n	c	a) n < $7A Cambio di controllo n = numero del controllore c = valore del controllore	4
			b) n = $7A Controllo locale on (c = $7F) off (c = 0)	5
			c) n = $7B, c = 0 Tutte le note off	6
			d) n = $7C, c = 0 Modo omni off (tutte le note off)	6
			e) n = $7D, c = 0 Modo omni on (tutte le note off)	6
			f) n = $7E Modo mono on (tutte le note off) c = numero di canali	6,7
			g) n = $7F, c = 0 Modo Poly on (tutte le note off)	6
$Cx	pp	-	Selezione programma "patch" pp = numero del programma	
$Dx	pr	-	Pressione sul canale pr = valore di pressione	3b
$Ex	l	m	Variazione della "ruota" per altezze l = 7 bit meno significativi m = 7 bit più significativi	8

Note

Se non altimenti specificato, i byte dei dati possono assumere un qualsiasi valore compreso tra 0 e $7F. La x rappresenta il numero di canale che occupa la cifra hex meno significativa del byte di stato (x = 0 corrisponde al canale 1, x = 0F al canale 16).

1) Il valore di p indica l'altezza della nota in semitoni. Il DO centrale corrisponde a p=$3C e tutti i DO corrispondono a valori multipli di $0C (12 decimale). Il valore di p, per una tastiera regolare da pianoforte con 88 tasti, varia dunque da $15 a $6C.
2) La velocità v può assumere valori compresi tra $01 e $7f. Nelle tastiere prive di sensori di velocità, il valore dovrebbe essere $40. Un messaggio di "nota on" con una velocità zero, equivale ad un messaggio di "nota off" con una velocità di $40. In genere questo è il metodo adottato con le tastiere più semplici, prive di senspri di rilascio, poiché consente di usufruire dello stato corrente dei messaggi di canale.
3) La "post-pressione" è la quantità di pressione operata sul tasto dopo che questo è stato premuto. Di solito viene impiegata per ottenere una modulazione senza dover intervenire sui parametri di modulazione. Esistono due tipi di sensori di pressione, pertanto sono previsti due diversi tipi di messaggio.
   a) La post-pressione individuale o polifonica implica la presenza di sensori di pressione su ciascun tasto ed ha effetto soltanto sulle note corrispondenti ai tasti premuti. Di conseguenza, oltre al valore della pressione, va inviato anche quello dell'altezza della nota (inoltre devono essere previsti circuiti di modulazione per ogni singola voce).
   b) La pressione di canale viene prodotta da un unico sensore di pressione che influenza, in modo identico, tutti i tasti. La pressione trasmessa corrisponde alla pressione massima (tutti i tasti premuti simultaneamente). L'effetto risultante riguarda tutte le voci.
4) Questi messaggi vengono inviati da unità di controllo esterne alle tastiere. Numeri di controllore, compresi tra $0 e $1f, corrispondono a controllori "continui" ed hanno un valore di controllo (c) tra $0 e $7F. In linea di principio queste unità di controllo assomigliano al joystick per computer basati su potenziometri (in pratica possono essere dispositivi quali gli stessi joystick, controlli a pedale o a fiato e via dicendo). I numeri di controllore compresi tra $20 e $3F sono eventualmente utilizzati per l'invio di ulteriori 7 bit meno significativi, destinati ai controllori da $0 a $1F, qualora occorra un'altissima definizione. I numeri di controllore compresi tra $40 e $5F corrispondono a unità di controllo basate su interruttori (quali pedali per sustain, portamento e via dicendo), in cui c'è posto a zero (off) oppure a $7F (on). Tali unità di controllo corrispondono ai normali joystick per computer dotati di interruttori. I numeri di controllore da $60 a $79 non sono definiti e quelli da $7A a $7F sono riservati ai messaggi di modo dei canali. Non esiste alcuna restrizione per l'abbinamento dei numeri di controllore a specifici dispositivi fisici; comunque è consuetudine associare alla ruota di modulazione il numero di controllore $1.
5) Il controllo locale viene usato, eventualmente, per interrompere il collegamento interno tra la normale unità d'immissione di uno strumento (ad esempio la tastiera) ed i circuiti generatori di suono, cosicché la tastiera, per esempio, invia dati soltanto all'uscita MIDI OUT ed il circuito generatore di suoni reagisce soltanto ai segnali applicati alla MIDI IN.
6) L'implementazione della condizione "tutte le note off" è opzionale e si consiglia di non utilizzare tali messaggi al posto dei comandi normali per l'annullamento di singole note. Tuttavia i fabbricanti non sembrano avere idee chiare in merito a questi messaggi, e ne rendono l'uso virtualmente inutile.
7) Il terzo byte del messaggio per il modo mono, specifica il numero totale di canali richiesto. Se questo numero è zero, il numero di canali eguaglia il numero di vodi disponibili sul ricevitore.
8) La "ruota" usata per modificare l'altezza delle note, è un dispositivo di controllo diverso da tutti gli altri poiché può produrre valori sia positivi che negativi. La posizione centrale equivale a l=$0 ed m=$40. Tuttavia molti ricevitori hanno soltanto 7 bit di risoluzione e reagiscono solo alle variazioni di m ignorando quelle di l.

MESSAGGI DI SISTEMA

Stato	Dato1	Dato2	Descrizione	Note
$F0	qualsiasi numero		Esclusivo di sistema	1
$F1			Non definito	
$F2	l	m	Puntatore alla posizione della media l = 7 bit meno significativi m = 7 bit più significativi	2
$F3	s		Selettore di melodia s = numero della melodia	
$F4			Non definito	
$F5			Non definito	
$F6			Richiesta di accordatura	3
$F7			Fine di esclusivo di sistema	1
$F8	-	-	Impulsi di sincronizzazione	4
$F9			Non definito	
$FA	-	-	Inizio	4
$FB	-	-	Continuazione	4
$FC	-	-	Arresto	4
$FD			Non definito	
$FE	-	-	Individuazione di attività	5
$FF	-	-	Reset del sistema	6

Note

I messaggi da $F8 a $FF sono quelli "in tempo reale" e possono essere inviati in qualsiasi momento (anche durante l'esecuzione di altri messaggi).

1) I messaggi esclusivi di sistema, che possono contenere una qualsiasi quantità di byte di dati, terminano con l'apposito codice di "fine messaggio esclusivo" ($F7) oppure con un qualsiasi altro byte di stato.
2) Questo messaggio viene utilizzato per predisporre arbitrariamente il puntatore al brano da eseguire. Questo puntatore è un registro interno che contiene il numero di battute (1 battuta = 6 impulsi di clock MIDI) a partire dall'inzio della sequenza (brano da eseguire).
3) Questo messaggio serve per richiedere l'accordatura degli oscillatori interni ai sintetizzatori analogici.
4) Questi messaggi vengono usati per sincronizzare l'unità sequenziatrice principale e quelle secondarie. Il temporizzatore di sistema viene regolato su una durata pari a 1/24 di un quarto di nota. Il comando "continuazione" differisce dal comando "inizio" in quanto l'esecuzione riprende dal valore corrente del puntatore al brano, anziché ripartire dal principio.
5) Il messaggio di "individuazione di attività" viene usato, all'occorrenza, come comando "di comodo" quando il sistema MIDI non svolge alcuna attività. Se utilizzato, occorre inviarlo in modo che non trascorrano più di 300ms senza lo svolgersi di una qualche attività. Se il ricevitore non riceve alcun segnale di "individuazione di attività", dovrebbe comportarsi normalmente ma, se lo riceve e trascorrono più di 300ms senza attività, il ricevitore dovrebbe spengere tutte le voci per poi tornare allo stato normale.
6) Questo messaggio viene utilizzato per inizializzare l'intero sistema subito dopo l'accensione. L'inizializzazione non può essere eseguita in modo automaticom all'accensione, altrimenti c'è il rischio che due o più componenti del sistema continuino ad inizializzarsi a vicenda all'infinito.

I messaggi che non compaiono nell'elenco qui riportato, non devono essere trasmessi, tanto più che non dovrebbero (almeno in teoria) essere riconosciuti dai ricevitori. In particolare occorre accertarsi che non vengano emessi messaggi spuri al momento dell'accensione.

Hi-Fi con la MIDI

Lo sviluppo di un programma che fa del computer un registratore in tempo reale

Quando vengono premuti i tasti di una tastiera MIDI, oppure viene azionato un dispositivo di controllo quale una manopola di regolazione della tonalità, i segnali generati vengono inviati alla presa MIDI OUT sotto forma di byte di dati. Il loro significato è già stato descritto in dettaglio, pertanto si può procedere alla realizzazione di un programma capace di ricevere dati MIDI e di elaborarli. Tuttavia il computer può essere usato anche per generare dati MIDI da trasmettere poi ad un sintetizzatore per attivarne i circuiti audio. Un esempio di questo secondo impiego si ha quando il computer viene utilizzato come sequenziatore introducendo, da tastiera, sequenze musicali composte secondo la tecnica "step-time" (ossia programmando ciascun passo all'interno di una battuta). Tale operazione viene eseguita prima che la trasmissione, attraverso la MIDI, attivi il sintetizzatore per l'esecuzione della linea melodica. In realtà questo compito è abbastanza semplice in quanto il sistema MIDI può eseguire operazioni ben più complesse. Si pensi, ad esempio, ad un sequenziatore che utilizza i 16 canali della MIDI per controllare una serie di tastiere e percussioni elettroniche!

Registrazione digitale

Lo schema sopra mostra come viene registrata una nota usando il programma per la registrazione digitale. Durante questa fase, il sintetizzatore trasmette un messaggio "nota on" (quando viene premuto un tasto) ed un messaggio "nota off" (quando il tasto viene rilasciato). I messaggi vengono immagazzinati in memoria insieme all'informazione di pausa che indica l'intervallo di tempo, espresso in unità di 2 ms, tra un evento ed il successivo. Nell'esempio, l'intervallo tra i messaggi di nota on ed off, viene registrato nei due byte $F0 e $37. Durante la fase di riproduzione il programma utilizza l'informazione di pausa per ritardare la trasmissione dei messaggi di nota e riprodurre il brano musicale con la massima "fedeltà".

Riuscendo a sviluppare un programma capace di registrare, nella memoria del computer in tempo reale, una melodia suonata su di un sintetizzatore, è possibile combinare alcune caratteristiche delle diverse modalità d'impiego del computer per interagire con gli strumenti collegati alla MIDI. Successivamente il programma, in risposta ad opportune istruzioni, ritrasmette la melodia al sintetizzatore che la riproduce tramite la MIDI. Naturalmente la tecnica di registrazione è digitale.

Sebbene apparentemente simile a quella analogica usata dai registratori a nastro magnetico, la registrazione digitale, attraverso la MIDI, si basa su principi assai diversi. Quando viene eseguita una registrazione da nastro, i suoni musicali vengono codificati sotto forma di configurazioni magnetiche sulla superficie del nastro: la loro posizione sul nastro determina la sequenza di riproduzione dei suoni e le pause tra un suono e l'altro. Facciamo un esempio: due note registrate con questa tecnica e separate da una pausa di due secondi.

Durante tale intervallo il nastro continua a scorrere sotto le testine di registrazione anche se non viene memorizzato alcun suono. Al momento che il nastro viene “riletto”, la lunghezza del segmento non registrato determina la pausa tra l’emissione della prima e della seconda nota.
In un sistema a registrazione digitale abbinato al MIDI, le due note corrispondono alla pressione di due tasti, ciascuno dei quali genera un messaggio MIDI. Una procedura di registrazione in tempo reale deve essere in grado di permettere al computer di registrare la pausa che intercorre tra la ricezione del primo e del secondo messaggio MIDI. I due messaggi, seguendo l’esempio della registrazione su nastro, potrebbero essere depositati in memoria utilizzando una matrice mentre, l’intervallo che li separa, si potrebbe rappresentare scrivendo valori “zero” in una serie di byte, ciascuno dei quali corrispondente ad una singola unità di tempo. Questa tecnica di registrazione, in tempo reale dei messaggi MIDI, comporta due risultati importanti: consente di riprodurre le due note nel momento stesso dell’emissione, conservando il corretto intervallo di tempo, e di risparmiare notevoli quantità di memoria.
Un metodo alternativo al precedente è quello di registrare l’informazione, relativa alla pausa, scrivendone il valore direttamente in un byte senza inserire alcun byte nullo. Se per esempio l’intervallo di tempo tra le due note fosse di 50 unità di tempo, sarebbe sufficiente introdurre nella matrice un byte di pausa contenente il valore 50 al posto di 50 byte “zero”.
Se da un lato questo sistema consente un apprezzabilissimo risparmio di memoria, dall’altro rende più difficoltoso prevedere l’occupazione in memoria di una sequenza. Indicativamente, a parità di durata, i brani musicali, caratterizzati da una rapida successione di note, richiedono un maggior numero di messaggi “on-off” per le note (e pertanto più byte) che non brani con una minore concentrazione di variazioni tonali. Il frequente ricorso al modificatore di tonalità genera ulteriori messaggi MIDI che contribuiscono ad accrescere il consumo di memoria.

Programma di registrazione digitale

Il programma alla fine di questo capitolo utilizza un particolare tipo di contatore (una variabile chiamata TEMP) per registrare gli intervalli di tempo tra due eventi successivi. La precisa misurazione delle pause e, di conseguenza, la “fedeltà” della registrazione, dipendono dal ritmo con il quale varia il contatore. La scelta della frequenza d’incremento ottimale è il risultato di un compromesso: nel programma qui presentato è stato fissato un intervallo di circa 2 ms; non tanto piccolo da risultare minore del tempo d’esecuzione delle routine ma neppure così grande da influire sulla precisione della registrazione. Questo intervallo di tempo (detto “risoluzione” della registrazione poiché rappresenta il più piccolo intervallo di tempo registrabile), viene generato da un temporizzatore situato in uno dei chip CIA o VIA del micro.
Il formato del messaggio MIDI deve essere leggermente ampliato per tener conto della temporizzazione; quindi, prima di ogni messaggio MIDI, viene inserito un byte che rappresenta il numero di intervalli di tempo trascorsi dall’ultimo messaggio. Questo byte può registrare un massimo di 239 unità di tempo, ossia valori compresi tra $0 e $EF.
Se dopo 239 intervalli il computer non ha ricevuto un nuovo messaggio, viene registrato un errore di “overflow” (superamento della capacità) rappresentato dal byte $F0. Infine è stato scelto un messaggio a singolo byte, $FF, per segnalare la fine della sequenza. I valori $F1 e $FE non vengono utilizzati e sono a disposizione dell’utente.

PROGRAMMA DI REGISTRAZIONE DIGITALE

Caricatore BASIC

```
 10 FOR I=49152 TO 49581
 20 READ X: POKE I,X: S=S+X: NEXT
 30 READ X: IF S=X THEN PRINT “OK! DIGITARE SYS 49152”: END
 40 PRINT “ERRORE DI CHECKSUM!”
 50 END
100 DATA 76,7,192,2,83,201,241,169,3
110 DATA 141,0,222,169,21,141,0,222,169
120 DATA 255,141,173,193,169,0,141,6
130 DATA 220,169,8,141,7,220,169,17,141
140 DATA 15,220,32,228,255,201,0,240
150 DATA 249,201,69,208,1,96,201,82,240
160 DATA 4,201,80,208,236,72,120,169
170 DATA 127,141,0,220,169,173,133,247
180 DATA 169,193,133,248,169,173,141,5
190 DATA 192,169,241,141,6,192,160,0
200 DATA 104,201,82,8,240,66,169,2,32
210 DATA 69,193,32,44,193,201,255,208,4
220 DATA 0,76,6,193,141,4,192,201,240
230 DATA 173,4,192,240,12,32,246,192
240 DATA 240,251,206,4,192,208,246,176
250 DATA 223,32,44,193,72,173,0,222,41
260 DATA2,240,249,104,141,1,222,16,3
270 DATA 32,53,193,202,208,233,174,3
280 DATA 192,16,195,169,1,21,69,193,140
290 DATA 4,192,162,255,32,246,192,240
300 DATA 16,238,4,192,173,4,192,201,240
310 DATA 144,6,32,21,193,140,4,192,173
320 DATA0,222,41,1,208,6,224,1,48,224
330 DATA16,243,173,1,222,48,11,224,0
340 DATA 48,213,208,27,174,3,192,16,11
350 DATA 201,248,176,223,201,240,176
360 DATA 196,32,53,193,72,173,4,192,32
370 DATA 21,193,140,4,192,104,32,21,193
380 DATA 202,240,178,208,197,173,1,220
390 DATA 73,239,208,18,104,104,40,208,4
400 DATA 169,255,145,247,88,169,0,32,69
410 DATA 193,76,37,192,173,13,220,41,2
420 DATA 96,145,247,238,5,192,208,18
430 DATA 238,6,192,208,13,104,104,104
440 DATA 169,3,32,69,193,76,6,193,177
450 DATA 247,230,247,208,2,230,248,96
460 DATA 162,2,201,192,144,5,201,224
470 DATA 176,1,202,142,3,192,232,96,10
480 DATA 170,189,96,193,133,249,189,97
490 DATA 193,133,250,160,0,177,249,240
500 DATA 6,32,210,255,200,208,246,160,0
510 DATA 96,104,193,134,193,146,193,158
520 DATA 193,82,32,61,32,82,69,67,79,82
530 DATA 68,44,80,32,61,32,80,76,65,89
540 DATA 44,69,32,61,32,69,88,73,84,13
550 DATA 0,147,82,69,67,79,82,68,73,78
560 DATA 71,13,0,147,80,76,65,89,66,65
570 DATA 67,75,32,13,0,79,85,84,32,79
580 DATA 0,32,77,69,77,79,82,89,13,0
590 DATA 240
600 DATA 52178: REM * CHECKSUM *
```

Listato Assembly

```
        *          = $C000
        IMMISS     = $FFE4      ; accetta un carattere
        OUTCAR     = $FFD2      ; visualizza un carattere
        JMP INIZIO
        NBYTE *    = *+1        ; n. byte nel messaggio corrente
        TEMP *     = *+1        ; pausa in periodi di 2 ms

        MEMLIB *   = *+2        ; n. di byte liberi
        DATAREG    = $DE01      ; reg. di trasm./ricez. su ACIA
        STAREG     = DE00       ; reg. di stato/controllo
        ARRESTO    = $EF
        MEM        = $F7        ; puntat. alla memoria dei dati
        PUNT       = F9         ; punt. alla stringa letta
        START      = *
; ++++ PREPARA LA  ROUTINE ++++
        LDA #$03
        STA STAREG              ; inizializza il chip 6850
        LDA #$16
        STA STAREG
G10     LDA #$FF
        STA INIMEM              ; azzeramento iniziale
        LDA #$00
        STA $DC06
        LDA #$08
        STA $DC07
        LDA #$11
        STA $DC0F               ; avvia il temporizzatore B
G20     JSR IMMISS
        CMP #$00
        BEQ G20
        CMP #69                 ;premere 'E' per uscire
        BNE G22
        RTS
G22     = *                     ; uscita non richiesta
        CMP #82
        BEQ G30
        CMP #80
        BNE G20
; +++ INIZIO PROGRAMMA IN TEMPO REALE +++
G30     PHA                     ; salva comando di registr./riproduz.
        SEI
        LDA #$7F
        STA $DC00               ; esplora l'ultima linea di tasti
        LDA #<INIMEM
        STA MEM
        LDA #>INIMEM
        STA MEM+1
        STA #<BYTELIB
        STA MEMLIB
        LDA #>BYTELIB
        STA MEMLIB+1
        LDY #$00
        PLA
        CMP #82
        PHP                     ; salva il flag Z=registrazione
        BEQ R00
                                ;
P00     = *                     ; esegue la routine
                                ;
        LDA #$02
        JSR STASTR
P05     JSR READ
        CMP #$FF
        BNE P07                 ; esce se arrivato alla fine
        PLP                     ; aggiusta lo stack
        JMP C20                 ; procedura d'uscita
P07     STA TEMP
        CMP #$F0                ; valore del Carry=£f0 legge
        LDA TEMP
        BEQ P30                 ; salta se non deve attendere
P10     JSR VERIF
        BEQ P10
        DEC TEMP
        BNE P10                 ; salta se tempo non scaduto
        BCS P05                 ; se $F0, legge pausa successiva
                                ;
P30     = *                     ; legge messagbio (byte) successivo
        JSR READ
        PHA                     ; salva il byte di dati
                                ;
P35     = *                     ; attende il base al registro TX
        LDA STAREG
        AND #$02
        BEQ P35                 ; salta se ancora pieno
                                ;
        PLA                     ; accetta dati
        STA DATAREG             ; li invia alla MIDI
        BPL P50                 ; salta se non è byte di stato
        JSR GETNO               ; accetta numero dei byte di dati
P50     DEX                     ; decrementa il contatore di byte
        BNE P30                 ; riceve il byte di dati seguente
        LDX NBYTE
        BPL P05                 ; riceve il byte di clock
                                ;
R00     = *                     ; routine di registrazione
                                ;
        LDA #$01
        JSR STASTR
        STY TEMP                ; TEMP = 0
R05     LDX #$FF                ; inizializza a nessun dato
R10     JSR VERIF
        BEQ R20
        INC TEMP
        LDA TEMP
        CMP #$F0
        BCC R20
        JSR MEMOR
        STY TEMP
                                ;
R20     = *                     ; cerca dati MIDI
        LDA STAREG
        AND #$01
        BNE R40                 ; salta se trova dei dati
        CPX #$01
        BMI R10                 ; salta se trova una pausa
        BPL R20                 ; salta se trova messaggio MIDI
                                ;
R40     = *                     ; accetta il dato
        LDA DATAREG
        BMI R50                 ; salta se trova byte di stato
        CPX #$00
        BMI R10                 ; salta se manca informaz. canale
        BNE R80                 ; salta se trova messaggio MIDI
        LDX NBYTE               ; azzera il contatore di byte
        BPL R60
                                ;
R50     = *                     ; elabora il byte di stato
        CMP #$F8
        BCS R20                 ; ignora se sist. in tempo reale
        CMP #$F0
        BCS R05                 ; salta se manca stato del canale
        JSR GETNO               ; accetta il n. dei byte di dati
                                ;
R60     = *                     ; registra il dato
        PHA
        LDA TEMP
        JSR MEMOR
        STY TEMP                ; azzera il contatore
        PLA
                                ;
R80     = *                     ; registra il dato MIDI
        JSR MEMOR
        DEX
        BEQ R10
        BNE R20
                                ;
VERIF   = *
                                ;
        LDA $DC01
        EOR #ARRESTO            ; lascia inalterato il Carry
        BNE C40
        PLA
        PLA                     ; aggiusta lo stack
        PLP                     ; esamina comando registr./riproduz.
        BNE C20                 ; salta se registrazione
        LDA #$FF
        STA (MEM),Y
C20     = *
        CLI
        LDA #$00
        JSR STASTR
        JMP G20                 ; attende il comando successivo
                                ;
C40     = *                     ; esamina il temporizzatore B
        LDA $DC0D
        AND #$02                ; flag Z = tempo non scaduto
        RTS
                                ;
MEMOR   = *                     ; registra il dato in memoria
                                ;
        STA (MEM),Y
        INC MEMLIB              ; calcola la memoria libera
        BNE AGGPUNT
        INC MEMLIB+1
        BNE AGGPUNT
        PLA
        PLA
        PLA
        LDA #$03                ; messaggio d'errore
        JSR STASTR
        JMP C20                 ; il programma riparte da capo
                                ;
READ    = *                     ; legge un dato dalla memoria
                                ;
        LDA (MEM),Y
                                ;
AGGPUNT = *                     ; aggiorna puntatore alla memoria
                                ;
```

```
         INC MEM
         BNE P20
         INC MEM+1
P20      RTS
                             ;
GETNO    = *                 ; riceve in acc. il n. dei byte di dati
                             ; per lo stato
                             ;
         LDX #$02            ; inizializza a 2 byte di dati
         CMP #$C0
         BCC N10
         CMP #$E0
         BCS N10
         DEX                 ; 1 byte di dati se C0<=A<E0
N10      STX NBYTE
         INX                 ; aggiunge 1 per lo stato
         RTS
                             ;
STASTR   = *                 ; visualizza una stringa
                             ;
         ASL A
         TAX
         LDA TABMESS,X
         STA PUNT
         LDA TABMESS+1,X
         STA PUNT+1
         LDY #$00
M10      LDA (PUNT),Y
         BEQ M70
         JSR OUTCAR
         INY
         BNE M10
M70      LDY #$00
         RTS
                             ;
                             ;
TABMESS = *                  ; tabella dei messaggi
                             ;
         .WORD    MESS0
         .WORD    MESS1
         .WORD    MESS2
         .WORD    MESS3
                             ;
MESS0    .BYTE ‘R = REGISTRAZIONE, P = RIPRODUZIONE, E = USCITA’,
                  $0D,0
MESS1    .BYTE 147, ‘REGISTRAZIONE’, $0D,0
MESS2    .BYTE 147, ‘RIPRODUZIONE’, $0D,0
MESS3    .BYTE ‘MEMORIA PIENA’, $0D,0
INIMEM   = *                 ; inizio memoria per dati
BYTELIB  = INIMEM - $D000
```

Funzionamento del programma

Il programma di registrazione digitale si basa su un algoritmo molto semplice. Quando viene selezionata l’opzione “registrazione”, il programma registra i messaggi in arrivo presenti sulla porta MIDI IN. La registrazione si interrompe quando la memoria disponibile è piena oppure quando viene premuta la barra spaziatrice. L’opzione “riproduzione” ritrasmette la sequenza registrata attraverso la porta MIDI OUT. Se non viene premuta la barra spaziatrice, l’emissione prosegue fino al termine della sequenza.

Una eventuale nuova registrazione cancella quella precedente.